1984-06

Implementation of an Intel 8086 microprocessor-based realization library for the Control System Design Language

Cetel, Alan Jeffrey

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

IMPLEMENTATION OF AN INTEL 8086
MICROPROCESSOR-BASED REALIZATION LIBRARY
FOR THE CONTROL SYSTEM DESIGN LANGUAGE

by

Alan Jeffrey Cetel

June 1984

Thesis Advisor: H. H. Loomis, Jr.

Approved for public release; distribution unlimited

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS
BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
Implementation of an Intel 8086 Microprocessor-based Realization Library for the Control System Design Language

5. TYPE OF REPORT & PERIOD COVERED
Master's Thesis;
June 1984

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Alan Jeffrey Cetel

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Naval Postgraduate School
Monterey, California 93943

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Naval Postgraduate School
Monterey, California 93943

12. REPORT DATE
June 1984

13. NUMBER OF PAGES
170

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Computer Aided Design; Microprocessor; 8086; Control Systems Design; Assembly Language Library

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
A library for use by the computer aided design system, known as the Control System Design Environment, made up of hardware and software primitives of the Intel 8086 microprocessor family, was written to extend the capabilities of the design system to more than two microprocessor families. Compatibility between this library and the Intel 8086 library was desired and achieved by use of designs originally realized with the 8080 library.

Implementation of an Intel 8086 Microprocessor-based Realization Library for the Control System Design Language

by

Alan Jeffrey Cetel
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1975

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1984

ABSTRACT

A library for use by the computer aided design system, known as the Control System Design Environment, made up of hardware and software primitives of the Intel 8086 microprocessor family, was written to extend the capabilities of the design system to more than two microprocessor families. Compatibility between this library and the Intel 8080 library was desired and achieved by use of designs originally realized with the 8080 library.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Herschel Loomis, Jr., and second reader, Lieutenant Colonel Alan Ross, for their guidance during this project.

I. INTRODUCTION

Computer Aided Design has seen an upsurge of use in the past decade, providing assistance in the design and eventual realization of everything from Very Large Scale Integration circuits to complete systems. The use of computers to simplify man's tasks is the chief motivating factor to develop more powerful computing devices. Recent trends in the development of Computer Aided Design (CAD) tools are toward a more 'user friendly' environment. Some CAD systems are so sophisticated that the user need not know the underlying principles of design in order to take a given concept, from its inception through actual realization in terms of either hardware and/or software. Starting with the simplest of descriptions, today's system or component designers may very well spend all his/her time designing at a computer keyboard or digitizing table.

What motivates the trend toward more efficient design systems are threefold: 1) greater speed in producing a viable design, 2) better use of scarce design system resources, and 3) decreased cost to produce the desired item. With the introduction of the microprocessor, it became apparent that its uses were limitless. Any conceivable electronic application spurred its use. From general purpose computing to specialized repetitive

functions the microprocessor found its way into industrial, commercial and military applications. As more uses were found, the ability to design with these devices lagged behind the rising requests for these new designs. Since the designs are labor intensive and design engineering expertise is at a premium, the concept of computer aided design was born. This technique helps to pare down the time it takes to design a new system or component. Engineers were, of course, not the only item in short supply. Computers needed to aid in these designs were also a scarce commodity. This led to the improvement of the design tools to see a design to completion with the least amount of resources used (e.g. time, computer use etc) [Ref. 1]. Once these major impediments had been overcome, the ability for the computer to help optimize the design was at hand. Factors to be considered were to decrease the number of chips or silicon acreage, decrease the power requirements, and decrease the overall cost of the system.

The use of microprocessors for real time control is just one of the applications that is seen in today's electronic environment. As in the past, design of these systems are a time consuming and complex process. In order for digital computing systems to provide a design environment, an examination of just how a design engineer might approach the problem is justified. The designer must

rely on a certain amount of background knowledge of the problem that is presented prior to his attempting its design. The implementation of this knowledge may be by the use of a database of design rules. These rules would apply to both the hardware and software factors required to realize a given problem. An extension of this idea would be the creation of a sequential listing of all possible combinations of circuit devices or software tasks that may be necessary to construct a device and the tasking of the digital computer to maintain a running list of attributes to insure that all design criteria are being met in the generation of a design realization.

This thesis will provide a library of hardware and software primitives implemented using the Intel Corporation iAPX 86/10 (8086) 16-bit microprocessor and its family of support chips. Chapter II will discuss the origins of the design environment in which this library will be used and its current implementation. A description of the structure of the realization library is presented in Chapter III. Chapter IV introduces the 8086 microprocessor and discusses its memory organization, the basic hardware design of a system implemented in this library, and a complete discussion of the software configuration including the use of assemblers, control, and arithmetic processes. Chapter V outlines the testing of this library. Finally, Chapter VI offers some recommendations for system improvement, observations and conclusions.

II. BACKGROUND

The system model proposed by Matelan [Ref. 2] for the design of microprocessor based controllers contains a concept of computer aided design whereby the specifications are not initially linked to the type of technology that might be used to implement a system. This binding of the design to a particular hardware and software technology is performed only after several intermediate processes are executed. These intermediate functions build symbol and timing tables, and prepare a table of the individual primitive names required to realize the design. It is only after this, that an implementation technology is chosen. The technology is contained in a volume of hardware and software primitives called the realization library.

A. CONTROL SYSTEM DESIGN LANGUAGE

Matelan's proposal suggests the use of a new high level language which he developed, called the Control System Design Language (CSDL), to support the definition of a design. An environment to interpret the design language and provide the necessary design system supervision and analysis is shown in Figure 1 [Ref. 3].

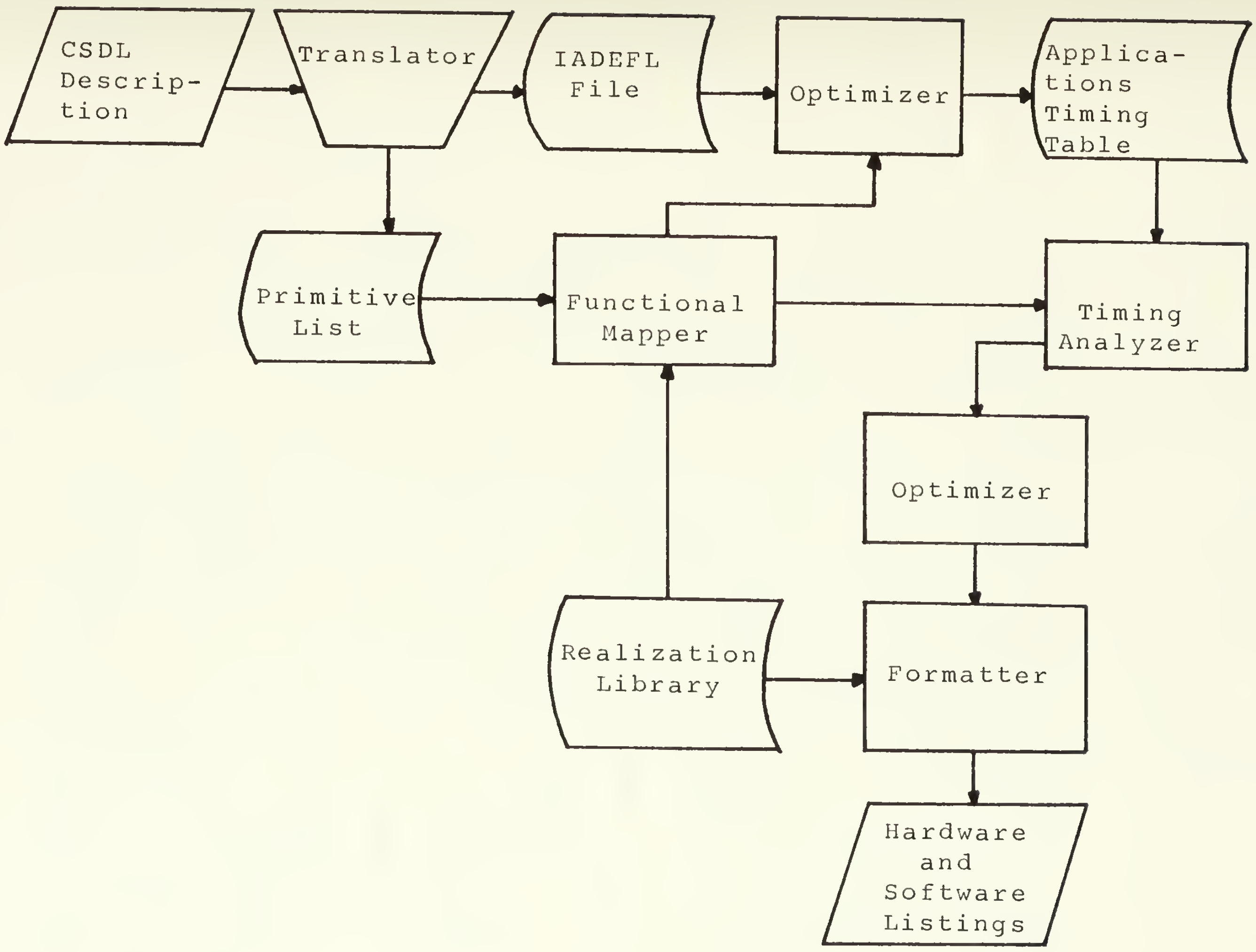


Figure 1. Flowchart of Design System

1. Design Environment

A CSDL description is written by the designer and is used as input to the design system. The translator decomposes the data from the description into two files: a list of primitives and a timing file. Examples and an explanation of these two files are presented later in this chapter.

The Optimizer module is first to receive these files. This module is tasked to control the overall execution of the system, serve as the input routine, and make a comparison of multiple realizations created by the system and choose the one that meets the criteria as specified by the designer. Following the optimizer is the Functional Mapper. It is the task of this module to ensure that the primitives required by the design are realized in the current realization library. Once the Functional Mapper successfully creates a listing of software titles, it becomes the task of the Timing Analyzer to ensure that all necessary timing constraints are satisfied. With a satisfactory timing analysis complete, the design is considered successful and the final function performed is the actual creation of the software and hardware listings. This process is accomplished by the Formatter. This module extracts from the current realization library all text that

is contained in each primitive in the primitive list and writes this text to the respective software and hardware output files.

CSDL defines functions and their timing constraints using the concept of contingency/task pairs. A contingency is defined as a function of an input variable or variables. The task associated with this contingency is executed only after the contingency is satisfied. Any given design then, must be stated in terms of its contingency/task pairs.

Matelan's Control System Design Language is used as the designer's interface to the overall design process. This language supports the input of a design by problem specification as follows: 1) an identification section, 2) an environment section, 3) a listing of contingency/task pairs, and 4) a procedures section.

The identification section is used as a header and record of when and by whom the design was created. The environment section describes the variables associated with the input and output ports of the device, their electrical characteristics, as well as the variables included in all computations internal to the software implementation. It is analogous to the variable declaration section of high level languages such as FORTRAN or PL/I. The contingency list describes the conditions that must be satisfied prior to the execution of its associated task. This list must also define the timing constraints that must be satisfied when

executing the contingency/task pair. Timing analysis is performed to insure that the time needed to execute a contingency and task falls within the required maximum time allowed by the designer prior to the execution of the next contingency/task pair. Finally, the procedures section contains the routines that make up each contingency/task.

a. Example Problem Description

An example CSDL listing is shown in Figure 2. The identification section is self-explanatory, containing the designers name, the date of file creation and the project name. The environment section contains all the input and output variable names and their bit length (precision). In this example the input variable X and output variable Y are both 16-bit values while the variable DL is a single bit. All of these variables are also identified as being TTL compatible. This section also contains the variable M and it's 16-bits that is used internally to the design. These variable descriptions are contained in the Arithmetic portion of the environment section. The function EXAMPLE that is contained in the contingency list is executed every 10 milliseconds, and when found to be true, the task RELIZE is performed. The final section of this high level description of the problem is the procedures section. It is in this section that the contingency/task pairs are explicitly defined. The contingency EXAMPLE is performed by sensing an external

```
IDENTIFICATION

        DESIGNER:  "A. J. CETEL"
        DATE:  "14 MARCH 1984"
        PROJECT:  "DESIGN EXAMPLE"

ENVIRONMENT

        INPUT:  X,16,TTL;DL,1,TTL;INAME,1,TTL;
        OUTPUT:  Y,16,TTL;
        ARITHMETIC:  M,16;

PROCEDURES

        FUNCTION EXAMPLE:
                EXAMPLE := 0:
                SENSE (INAME);
                IF INAME = 1 THEN EXAMPLE:= 1; END IF;
        END EXAMPLE ;

        TASK RELIZE;
                SENSE (X);
                Y := ( X * 10 ) + M;
                ISSUE (Y);
        END RELIZE;


CONTINGENCY LIST
        WHEN EXAMPLE : 10MS DO RELIZE;
END
```

Figure 2. CSDL Design Example

variable INAME. If the value of this variable is equal to one, then the value of EXAMPLE is also set to one. With the value of EXAMPLE equal to one, the contingency is satisfied and the task RELIZE is performed. The task RELIZE is a simple arithmetic expression that senses the value of X, then computes the value of Y using variables, as described in the arithmetic portion of the environment section, as well as any constants required, and latches or otherwise makes the value of the variable Y available at a TTL compatible output port. Had the variable INAME been another value, then EXAMPLE would not be set to one, the contingency therefore would not be satisfied and the task RELIZE would not be performed.

B. CURRENT IMPLEMENTATION

Implementation of Matelan's design system was produced as part of a dissertation by Ross [Ref. 4]. This implementation however, does not support the input translator to the system. A CSDL compiler (translator), designed to be the input module, implemented in Pascal, is under development by Carson [Ref. 5].

The description of the input specification in the previous section is presented for continuity purposes only. In order to relize a design as the system is currently implemented, a number of intermediate files are required.

1. Primitive File

A list of primitives, implemented as the file PRIMITIVE.DAT, contains a representation of all the required primitives to realize a given system. It contains each primitive to be extracted from a realization volume and the information required to generate the hardware and software listings. Variable names, data constants, and data size descriptions are all contained in the primitive listing. The order of primitives is directly related to the order required to implement the device. An example of this file is given in Figure 3. Each line in the primitive file is a sequential list of the lines to be taken from a realization library required to fulfill given design requirements.

A system primitive list is always required to initialize pointers, include software heading or assembly language directives, and call the processor primitives required to eventually realize the design. This item is shown in the first group of primitives labeled - t.generated for: system. All of the above requirements are accomplished within the primitive s.main(::). All arguements passed to the primitive are included between the parenthesis and before the first colon. The precisions of any variables passed are contained between the two colons. No information is needed after the second colon, however the second colon is required due to the strict format of Ross' implementation.

```
T.GENERATED FOR:  SYSTEM
s.main       (::)
T.GENERATED FOR:  EXAMPLE
s.proc       (EXAMPLE::)
s.eq         (@T1,INAME,@C001:8,1,1:)
s.jmpf       (@T1,@1000:8:)
s.assigncons (INAME,0:1,1:)
s.assigncons (EXAMPLE,1:1,1:)
s.loc        (@1000:8:)
s.exitproc   (EXAMPLE,0::)
s.cons       (@C001,0:1,1:)
s.var        (EXAMPLE:1:)
s.var        (INAME:1:)
s.var        (@T1:8:)
T.GENERATED FOR:  RELIZE
s.proc       (RELIZE::)
s.assigncons (INAME,0:1,1:)
s.anain      (X,-10,10,5:16:)
s.mul        (M1,X,C1:16,16,16:)
s.add        (Y,M1,M:16,16,16:)
s.anaout     (Y,-10,10:16:)
s.cons       (C1,10:16:)
```

Figure 3. Example Primitive List

The first contingency, EXAMPLE, requires a procedure beginning and end, shown by the software primitive s.proc and s.exitproc. The argument passed for this primitive is simply the contingency name. A primitive titled s.eq checks for equivalence between the one-bit variables labeled iname and @c001 and if true assigns the 8-bit variable @t1 the value of true (=1). The remaining primitives provide constant assignments, establishment of variables, arithmetic routines, and input and output software routines. Hardware primitives are usually called from their software counterparts.

It is from this file that the functional mapper selects a primitive from the realization library that matches not only the title, but also the number of arguments required and the precisions of these arguments.

2. Timing File

A timing file to be used in the analysis of each contingency/task pair must also be generated. This file, IADEFL.DAT, contains the timing constraints imposed by each contingency/task pair. This file is used by the timing analyzer, along with the previously built list of software titles, to determine if the time constraints imposed by the designer are met. Included in this file is the design criteria section, added by Ross to provide a metric by which to determine the optimal realization of a design. The designer may choose one of three criteria to produce a

design realization: 1) first realization that is generated 2) most inexpensive and 3) the realization with the least power consumption. A detailed description of this file is not included here since is not directly concerned with the construction of a realization library. It is mentioned to provide the reader with a better overview of the design environment in which the realization library exists.

III. REALIZATION LIBRARY

Ross' original implementation provided a library of hardware and software primitives using the Intel 8080 microprocessor. This library established the format required to implement subsequent microprocessor libraries. To date, the addition of a Zilog Z-80 library by Smith [Ref. 6], and this thesis, using the Intel 8086, are the only other libraries written. Chapter VI of this paper outlines a method to eliminate the need to write individual assembly code libraries for every microprocessor family.

A. LIBRARY FORMAT

The rigid structure of the realization library format requires the writer to build a library using one of ten possible formats for each line of the library. These formats are:

1) primitive title line
2) comment line
3) calculation line
4) attribute line
5) call line
6) include line
7) if line
8) begin text line
9) endtext line
10) text line

Each line of a library begins with a 'v' in column one followed by a four digit line number in columns two through five, and text in columns six through eighty. A line may not extend past column eighty in the current implementation. The first digit of the four digit line number specifies the volume number. If more than 999 lines are needed in a library or more than nine volumes are written, alphabetic characters can be used.

The first line of the library, line number vx000 (where the x is the volume number), identifies the microprocessor family (intel 8086 cpu), the clock period, any additional delays in accessing memory, and a monitor constant all used in the timing analysisto determine if a particular design is realizable. Each of these attributes are separated by colons (see the first line of the realization library, Appendix C). Following this line is the library index, which is a copy of every primitive title line contained in the library. Line numbers in the index are not consecutive, but are the actual line numbers of the primitive title lines' location in the body of the library. However, a counter tracks each line listed in the index so the first primitive title line in the body of the library contains its actual line location from the beginning of the library, including the index listing. The index of primitives is followed by an '.end index' line starting in column seven.

1. Primitive Title Line

Title lines define either hardware or software primitive types. A decomposition of a hardware and software title line are shown in Figures 4 and 5, respectively. The hardware title line starts with an 'h' in column 6, a period in column 7, and the primitive title in column 8 through 17. If the title is less than 10 characters in length, then blanks are inserted through column 17. Column 18 contains a left parenthesis followed by the parameters of the primitive. These parameters are listed in three variable length fields separated by colons. If any of these fields are empty, the colons must still appear. The first field specified the dummy argument names, followed by a colon, with the selection criteria in the second field. Argument names can be up to six characters in length and are separated by commas. The arguments must appear in the same order as the actual arguments in the primitive calls.

The selection criteria, in the second field, contains the minimum and maximum size, in terms of bit count or precision, of each argument. Each of these values are separated by commas. These values are used by the functional mapper to determine if the primitive realization meets the requirements of the primitive being invoked. This field is also followed by a colon.

The third field contains the attributes of the realization. For a hardware primitive, these attributes are

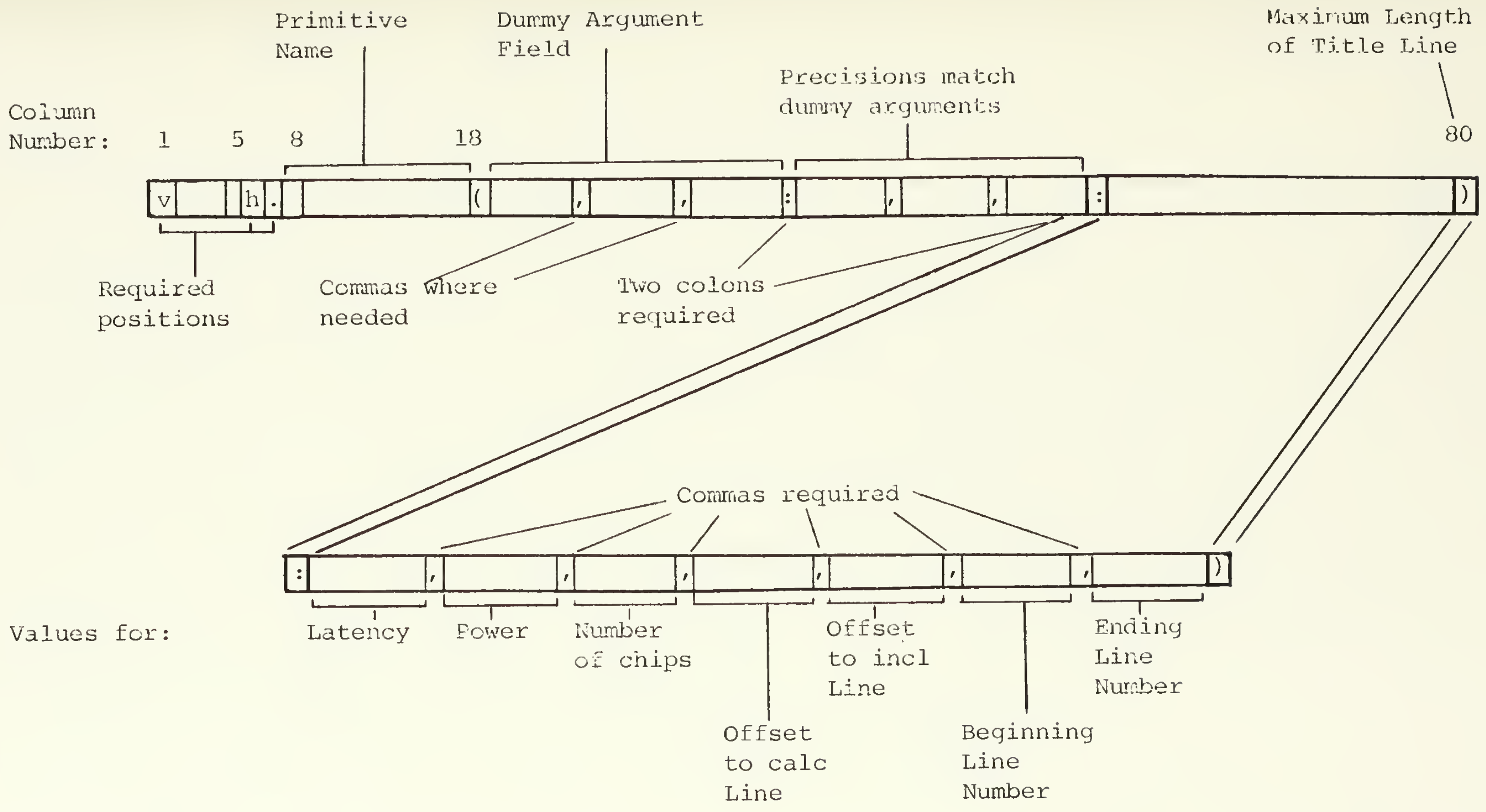


Figure 4. Example Hardware Primitive Title Line

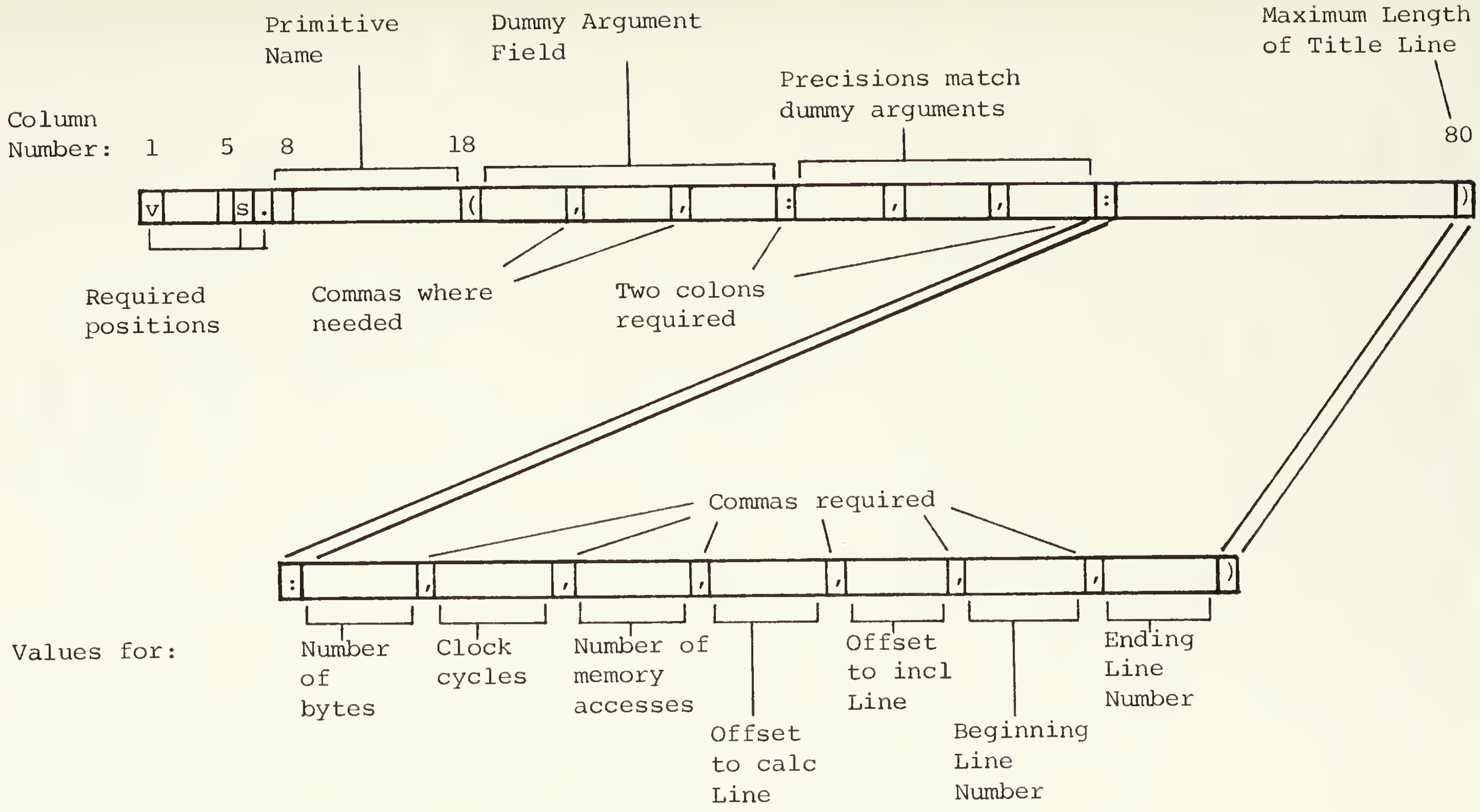


Figure 5. Example Software Primitive Title Line

(in order, separated by commas): 1) latency of the device (delay in accessing), 2) amount of power required by the electronic components contained in the primitive (in milliwatts), and 3) the number of chips contained in the primitive. For a software primitive, these three attributes are (again in order, separated by commas): 1) number of 8-bit bytes of storage required, 2) the number of clock cycles required to execute the primitive, and 3) the number of references to external memory made during the primitive.

Both hardware and software primitives use the next four attributes as a flag to indicate whether any calculation ('calc') or include ('incl') lines are contained in the primitive, and the line numbers of the first and last line of the primitive within the body of the library. If no 'calc' or 'incl' lines are contained in the primitive, a value of zero is given to these attributes. A primitive that contains a 'calc' or 'incl' line has a value inserted into this location that corresponds to the offset from the first line of the primitive to the line where a 'calc' or 'incl' does occur. If any of these attribute values depend on formal parameters passed by the calling primitive line, the attribute is given a negative sign in the title line. The offset to an attribute ('attr') line which calculates the actual value of the corresponding attribute in the absolute value of this negative attribute in the title line.

2. Comment Line

Each comment line begins in column 6 with 'com'. Columns 9 through 80 can be any desired text, numerals, or special characters. This line is ignored by the system and is therefore not written to any output file.

3. Calculation Line

Calculation or 'calc' lines are used in library primitives to manipulate system global variables. This line begins with 'calc' in columns 6 through 9. The list of available system globals and the definition of each is contained in Appendix B. This global list is a combination of Ross' original implementation and Pollock's [Ref. 7] additions for the Intel 8080 cpu, and the additions provided by this paper. This universal globals list is generated to provide the design environment with the ability to select between microprocessor families in order to selected the appropriate realization to meet timing and design criteria without the need for a separate globals list for each library. The 'calc' lines assign values to the global variables during the generation of the design. Calculations are performed using mathematic expressions with global names and dummy parameters as variables. Arithmetic operators available are +, -, *, -, and ** and any number of pairs of left and right parenthesis to force the order of execution of operators. The operators are interpreted in the same way as they would be in Fortran.

4. Attribute Line

Similar to the 'calc' line, attribute lines begin with 'attr' in columns 6 through 9 and are used to calculate the value of the attributes contained in the first three parameters of the third field of the primitive title line. For example, depending on the value of a certain global variable, a primitive may or may not add a new hardware component to the hardware realization. This means that a check would be required within the primitive to determine whether to include a component or not. The title line would show a chip count of zero, but following a check of a global for a certain value signifying the need for the addition of the component, the chip count would be increased by one using the 'attr' line (see Figure 6).

5. Call Line

This line is used to provide control within the library to invoke other primitives. As with other line items, this line begins with 'call' in columns 6 through 9. Prudent use of the 'call' line results in the reduction in the overall size of the library. A 'called' primitive's text is added to the output listing at the place where it is called.

6. Include Line

An include line is similar to the 'call' line with the difference being the included primitive text is placed at the end of the output listing after all of the other

Primitive with Flag

(Attributes computed within the primitive)

```
                      power    number of chips

                latency

h.invert     (in,out::0,0,0,0,0,1000,1010)
com primitive to define ttl invertor
if flgnme .ne. 1 skip 2                    Flag used to determine
attr pwr = pwr + 60                        if component is to be
attr chips = chips + 1                     added.

    (Body of primitive)
```

Equivalent Primitive with Attributes in Title Line

```
                      power    number of chips
               latency

h.invert     (in,out::0,60,1,0,0,1000,1007)
com primitive to define ttl invertor

    (Body of primitive)
```

Figure 6. Example Attribute Line Entry

primitives in the Formatter input primitive list are finished. This line begins with 'incl' in columns 6 through 9. The 'incl' line is used mainly to add hardware to the hardware output listing during the generation of the software output. For example, the addition of a hardware I/O port during the addition of the software text that supports the checking of information at that port address.

Both the 'incl' and 'call' lines must contain the actual arguments and selection criteria fields, if needed. Spacing requirements are the same as for the primitive title line and must be observed. Arguments must start in column 19 preceded by a left parenthesis in column 18 and must be separated by commas and followed by a colon. Another colon must follow the selection criteria field, if present, and the line must end with a right parenthesis.

7. <u>If Line</u>

Conditional branching is accomplished within the library by use of the 'if' line. This line begins with 'if' in columns 6 and 7 followed by a mathematic expression, a relational operator, .ne., .gt., .ge., .lt., or .le. (meanings as in Fortran), another mathematic expression, and a skip instruction to by-pass any number of lines forward or backward from the 'if' line within a primitive.. The format for this statement is:

```
if <math exp> <rel op> <math op> skip <# of lines>
```

If a backward skip is required a negative value is used for the "# of lines" argument. A skip backwards includes the 'if' line executing the branch and the line to be executed upon completion of the skip. This process is shown in Figure 7. An unconditional branch can be invoked by using the skip portion of an 'if' line.

8. Begin Text Line

This line precedes the actual lines of text that appear in the output listing and in combination with the 'endtext' line brackets the text of the primitive. The line begins with 'begin htext' for a hardware primitive, and 'begin stext' for a software primitive, in columns 6 through 16. No other characters appear on a begin text line.

9. Endtext Line

This line begins with 'endtext' in columns 6 through 12, for both hardware and software primitives, and in combination with 'begin text' marks the end of a text listing that is destined for an output file. Any number of 'begin text' and 'endtext' lines may appear in a primitive, but they must be used in pairs.

10. Text Line

A text line is free-form and allows the library writer to format lines of text between 'begin text' and 'endtext' in any way that will later be compatible as hardware and software output listings. These lines contain

		Number of Lines Skipped	
Line #		Forward	Backward
1	if mem .ge. ramptr skip 4	0	5
2	incl h.ram (::)	1	4
3	calc cnt = cnt + 1	2	3
4	calc mem = mem - 10	3	2
5	skip -5	4	1
6	begin stext		

Figure 7. Example If Line Entry

volume and line numbers, therefore the only restriction is to limit the text to columns 6 through 80. All text contained in text lines will be copied to the output listing exactly as written with two exceptions. Variables enclosed in pound signs (#) are interpreted as a call to a system procedure or function. The corresponding procedure will be implemented to generate character strings for the output listing. An example of this is the use of the #IDSEC# function in the primitive s.heading of Appendix C. This copies the lines of the identification table from the input file to the output listing. The other exception is the use of dummy arguments enclosed in brackets '<' and '>'. Each time a dummy argument is encountered in a text line enclosed in brackets, the current value of that argument is written to the output listing in the location specified in the text line.

IV. 8086 LIBRARY IMPLEMENTATION

The use of the Intel 8086 16-bit microprocessor is a logical extension of the currently available realization libraries to CSDL. Due to the similarities of the 8080 and Z-80 microprocessor architecture and instruction sets, a good comparison of execution speeds of a given design is easily done. In some cases, for example, a control system realization may not be possible with the slower 8080 cpu, but may easily be implemented by the Z-80 or 8086. The variables of cost and power consumption must also be taken into account to make a final decision as to the 'best' realization to be used (recall that these factors are part of the design criteria section of CSDL).

A. MEMORY ORGANIZATION

Figure 8 shows the standard memory map of the 8086. Memory organization used in the 8086 realization library is opposite from the previous two libraries. Whereas the 8080 and Z-80 libraries build the ROM (instruction) area from low to high memory locations and RAM (data and stack) area from high to low memory locations, the 8086 builds RAM from low to high with ROM stepping from high to low in 64K byte steps. Stack RAM is isolated from data RAM by use of the segment registers, discussed later in this chapter.

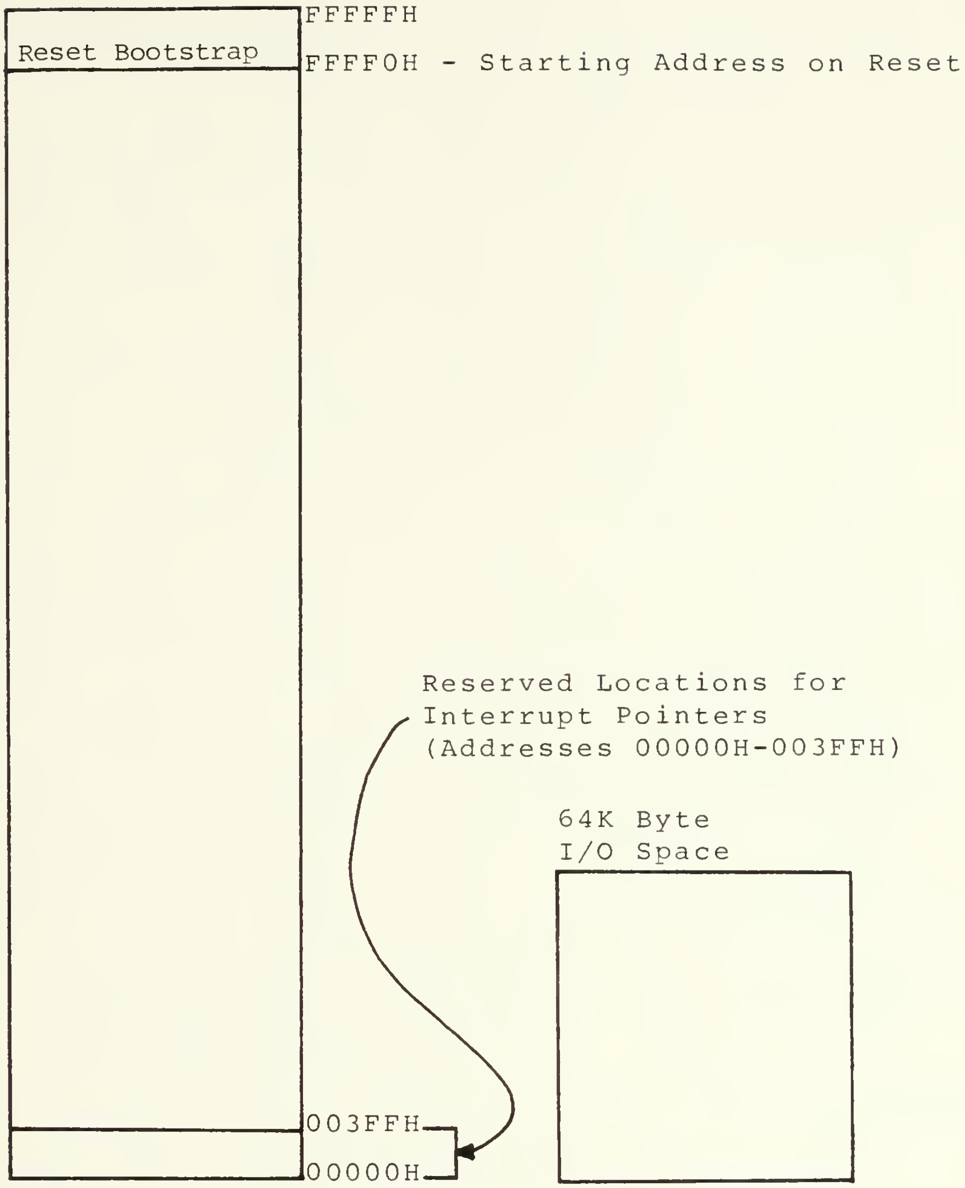


Figure 8. Standard 8086 Memory Organization

Realization library memory organization is illustrated in Figure 9. Stack memory occupies the lowest 1K of RAM followed by the data RAM which is allowed to increase in size up to the lower limit of the ROM area. This configuration chooses to take advantage of the instruction pointer (IP) being set to address FFFF0H upon microprocessor reset and the default value of the code segment register set to the highest 64K byte block of memory. Instructions are written into ROM by stepping to the lowest address of a 64K byte block and then incrementing addresses for each succeeding instruction. At the highest address location in any 64K byte block, a JUMP instruction is inserted to jump to the lowest address of the next lower 64K byte block of memory (see Figure 10). By keeping instructions in high memory, there is no overhead associated with manipulating segment register values that would be required to embed instructions in the middle of what would otherwise be data or stack memory addresses. Therefore, instruction ROM area remains in the higher memory locations. However, this method only allows for a code section 65516 bytes in length in this library's implementation. This is due to the inability of the design system to check for the current ROM pointer value during software file generation and evaluate the 'stuffing' of a jump instruction in the last three bytes of the current 64K byte memory block. A method to evaluate

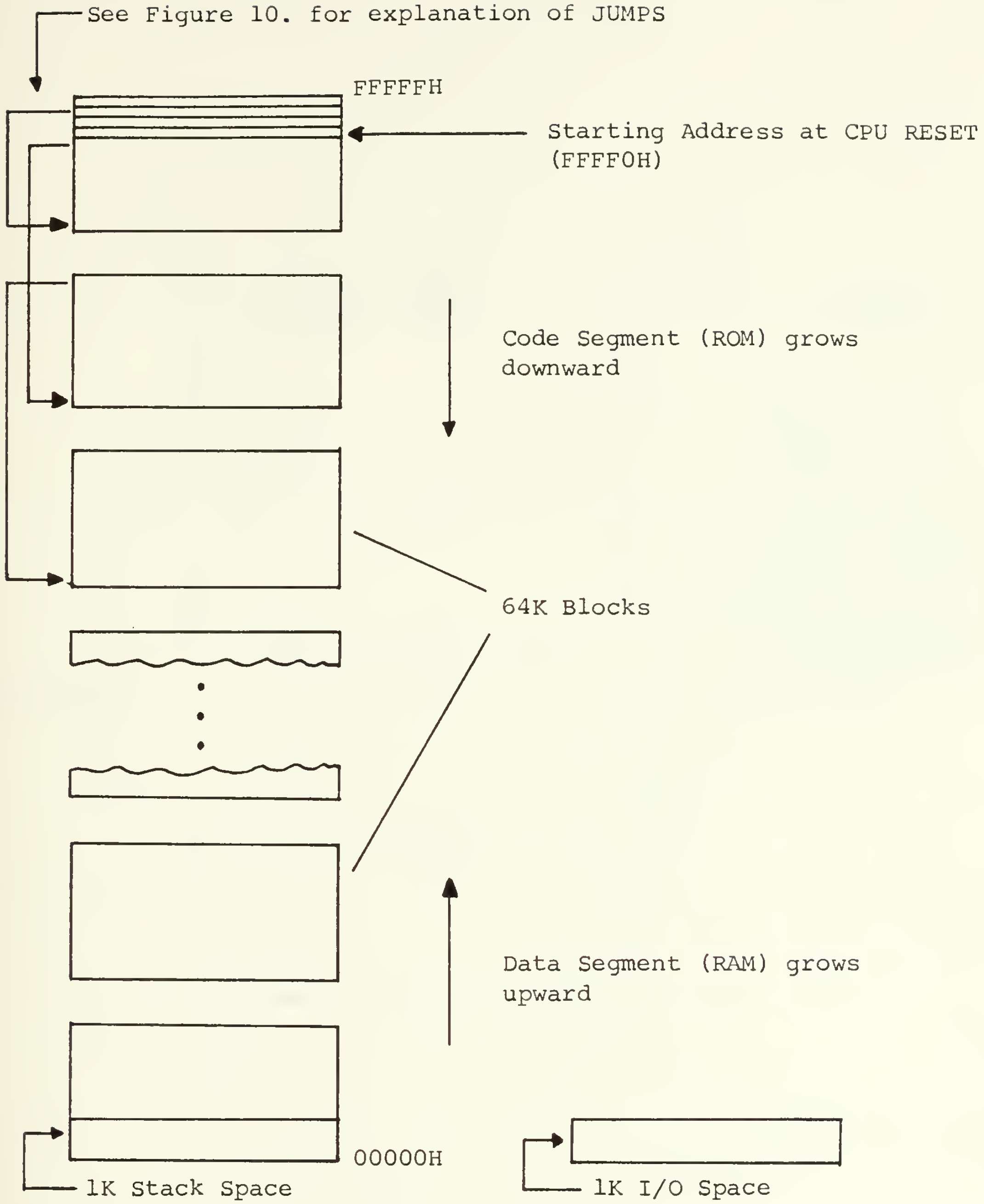

See Figure 10. for explanation of JUMPS
FFFFFH
Starting Address at CPU RESET (FFFF0H)
Code Segment (ROM) grows downward
64K Blocks
Data Segment (RAM) grows upward
00000H
1K Stack Space
1K I/O Space

Figure 9. Memory Organization in the 8086 Library

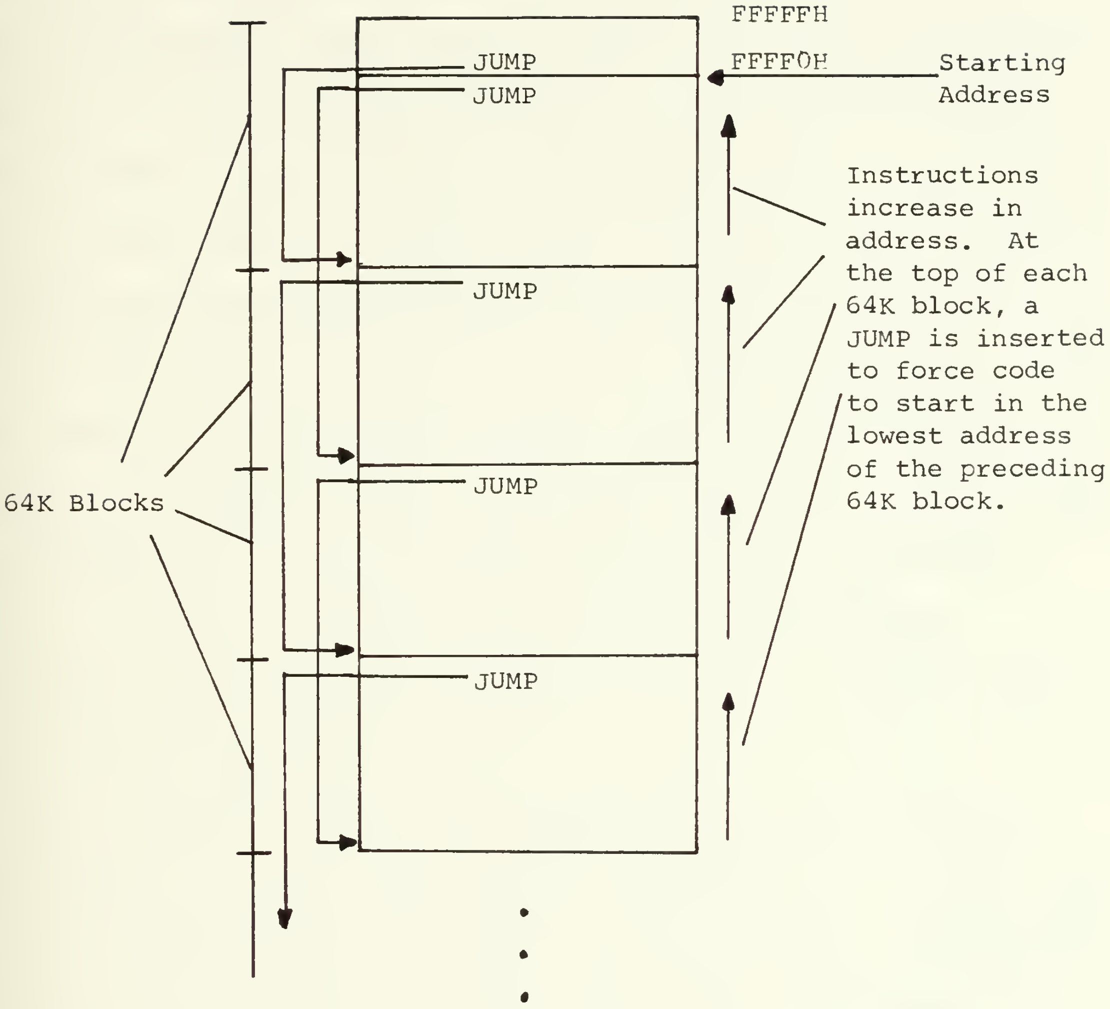

FFFFFH
FFFF0H
Starting Address
JUMP
JUMP
JUMP
JUMP
JUMP
64K Blocks
Instructions increase in address. At the top of each 64K block, a JUMP is inserted to force code to start in the lowest address of the preceding 64K block.

Figure 10. Insertion of JUMP Instructions

a primitive's assembly code line by line is necessary to execute this procedure, since a primitive may exceed a 64K byte memory boundary partially through that primitive's code. An alternative to this method would be to allow the system to delay the binding of ROM addresses until the entire instruction code has been generated. The appropriate number of 64K blocks of memory could then be generated for the hardware output listing. Once this is done, the address of the lowest block of memory may be inserted into the JUMP instruction located at FFFF0H using the system equates (e.g. sys14) found at the beginning of the assembly heading. This would then allow an uninterrupted, linear address space for use by the IP without the associated overhead of adding jump instructions at the high address of each 64K byte memory block. A third alternative would be to arbitrarily partition memory into two 512K byte segments, one for code and one for data/stack.

With program code in high memory, the stack and data RAM sections are put into the lower addresses. A stack area of 1K is established in the first 1K of memory. Immediately following the stack area is the data area. This area is allowed to grow to any size, up to the ROM area. For very large programs where data RAM and instruction ROM addresses approach each other, a scheme must be incorporated to ensure that neither of these areas will occupy a portion of the

same 16K byte block of memory. Figure 11 shows how this conflict is resolved. Current implementation does not consider this possible error condition.

1. Segment Registers

Addressing the one megabyte of memory available to an 8086-based system does not follow the standard microprocessor addressing convention. A 20-bit address bus is required to support the entire address space. Register structure within the 8086 is only 16-bits wide. To accomplish the construction of a 20-bit address, a second summer is used in conjunction with one of four segment registers. These segment registers define a base address within a 64K byte block of memory which allow code, data, and stack values to be separated in physical memory. The capability exists to fully or partially overlap segments within the physical memory space by storing the appropriate values in the respective segment registers. As is shown in Figure 9, the stack and data 64K byte segments overlap with the exception of the lowest 1K of memory. An "extra" segment can also be defined for whatever purpose the programmer deems necessary. This extra segment is not used in the construction of this library. Figure 12 illustrates how segment registers define base addresses. Figure 13 shows how these registers are used for computing an address. A 16-bit address in the IP, called the logical address, is

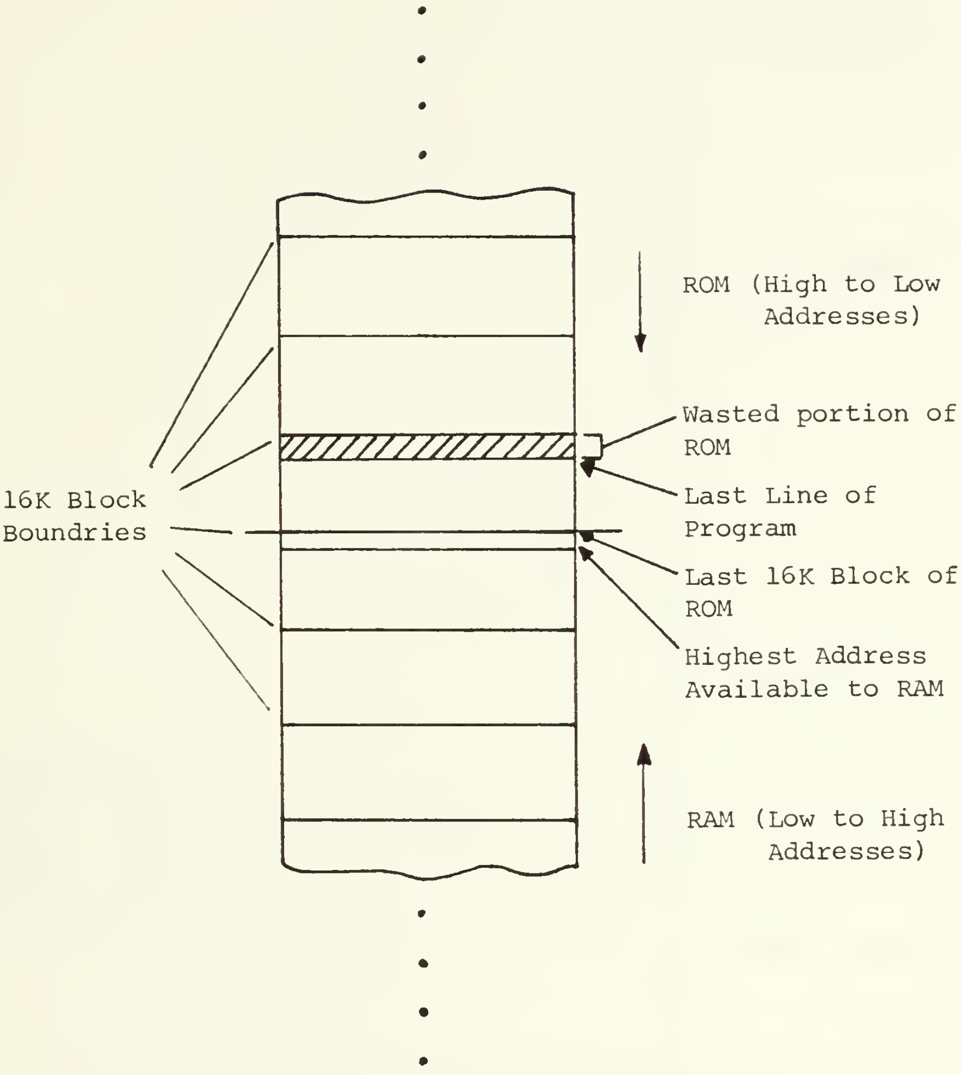

ROM (High to Low Addresses)
Wasted portion of ROM
16K Block Boundries
Last Line of Program
Last 16K Block of ROM
Highest Address Available to RAM
RAM (Low to High Addresses)

Figure 11. Overlapping of ROM and RAM Areas

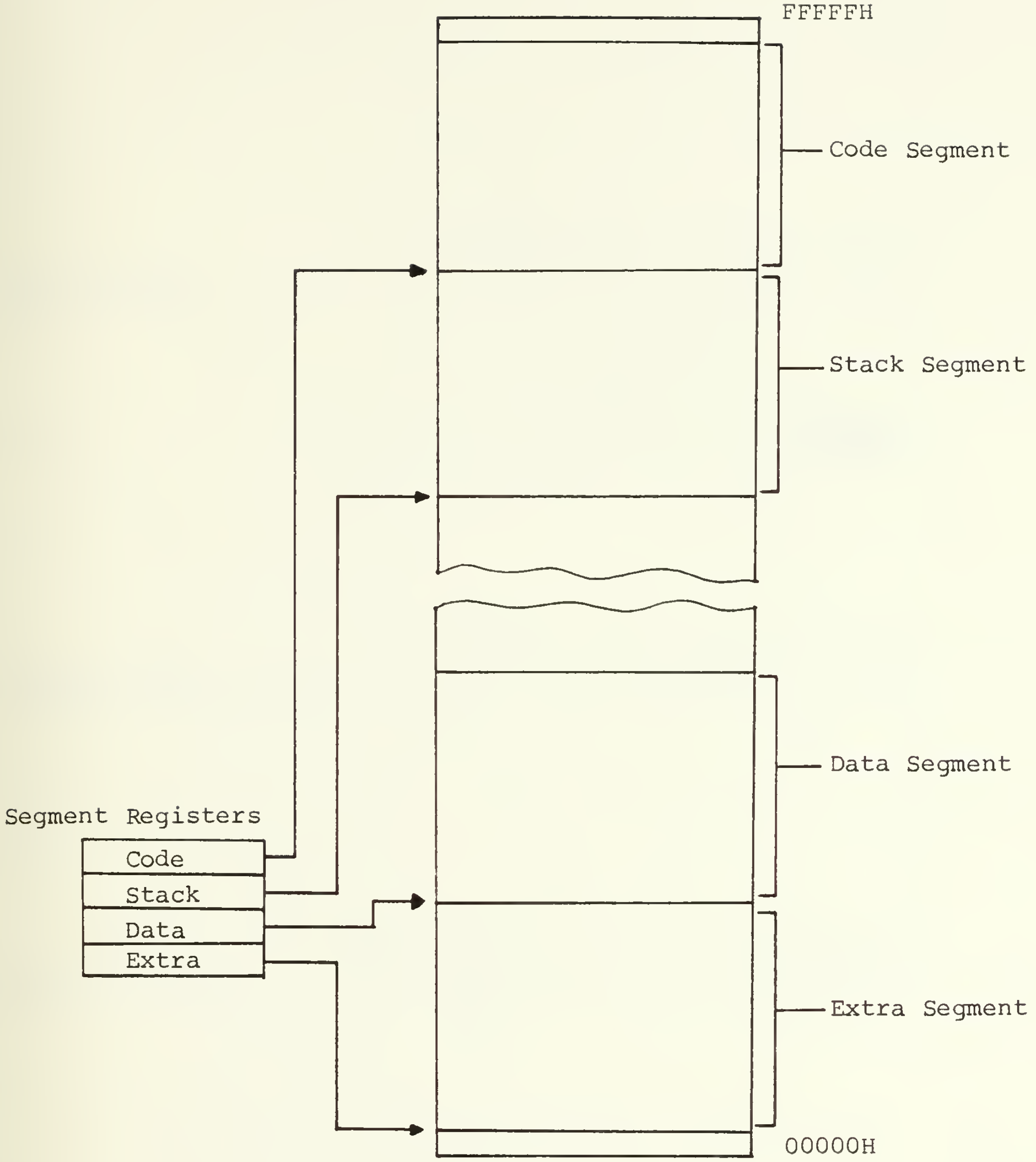


Figure 12. Definition of Base Addresses by Segment Registers

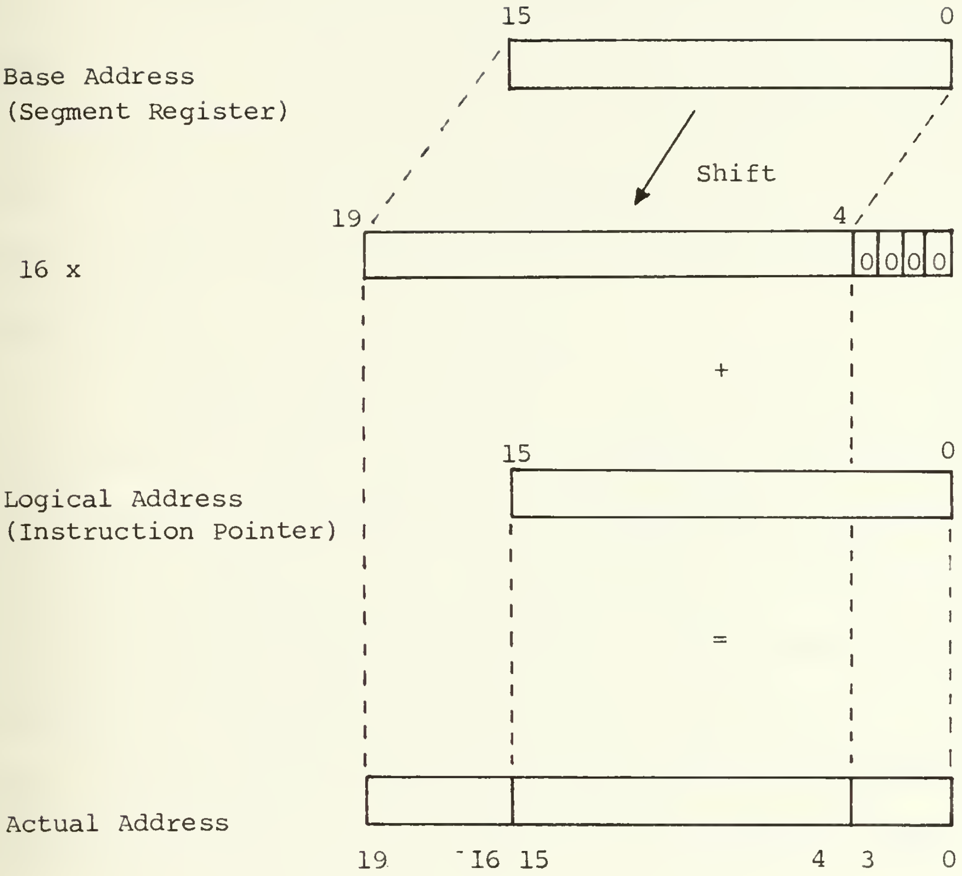


Figure 13. Constucting a 20-Bit Physical Address

added to the value contained in one of the segment registers, the base address. Prior to the addition, the base address is multiplied by sixteen, which effectively performs four left shifts and 'stuffs' zeros in the least significant nibble. The result of this addition produces a 20-bit real address that is subsequently placed on the address bus. On microprocessor reset, default values for the segment registers are loaded, but may be changed at any time by the program or upon assembly if the code, data, or stack regions extend beyond their respective 64K byte boundaries.

Changing the values of these registers within the primitives of this library is not done, with the exception of stack and data segment initialization. As program size increases, there exists a requirement to change the values in the segment registers as code or data cross 64K byte boundaries. The process of determining when to load a new base address into a segment register is transparent to the programmer, since this task is accomplished by the assembler.

2. Input/Output Memory

Input/Output memory is implemented in the 8086 as a separate 64K byte block of memory located in the lowest address space. In this implementation, I/O memory is restricted to the lowest 1K byte of memory. Although

arbitrarily chosen, this amount of I/O should accommodate even the most I/O oriented design. Any requirement to increase I/O memory can easily be accomplished by substituting a new base address for data RAM, thus allowing the stack to grow into the current data address space.

3. Interrupt Handling Considerations

Current implementation of CSDL does not support an interrupt driven, real-time controller. Therefore, the interrupt (INTR) and non-maskable interrupt (NMI) pins are tied to ground. Memory space that is normally reserved for interrupt handling in low memory of the 8086 is ignored and has been designated as the lower quarter of stack memory. If further research allows for interrupts in controller system generation, all that is required to alter the current memory map convention is to shift both the stack base address and the data base address values to an address area above the highest interrupt address. Without changing the data segment, a configuration allowing all 256 interrupts and a stack size of 768 bytes could be implemented by only shifting the stack base address to 256D.

B. OPERATING SYSTEM CONSIDERATIONS

The system monitor constructed from library primitives forms the operating system of the design realization (more on the monitor later). No outside intervention in the operation of controllers produced by this CAD system is

anticipated. Therefore, all memory is available for use by the code, data, and stack only.

C. HARDWARE DESCRIPTION

A large memory space combined with a high speed microprocessor makes even the simplest of hardware configurations generated from this library chip intensive. The composition of the library is based on individual IC's and their eventual integration onto a printed circuit or wire-wrap board. This follows the convention of the original 8080 library. It becomes possible then, to make a direct comparison between the two libraries when attempting to make a design criteria check. This differs from the prototyping scheme used by Smith in the generation of the Z-80 library.

1. 8086 Microprocessor and Support Chips

To allow for faster performance at the hardware level, the Intel 8086-2 8MHz microprocessor is used. Although a faster version, the 8086-1, is available which runs at 10MHz, the choice to remain at 8MHz is based on two requirements. First, and foremost, is memory compatibility. The option to obtain memory devices from other manufacturers was available, but for power requirements and overall system integrity, the use of Intel produced memory chips was made (discussion of memory follows later in this chapter). Secondly, the choice to proceed with the 8MHz version is

cost. In the long term, however, with the cost of most popular integrated circuitry decreasing, this point may be of little consequence.

Two modes of operation are available with the 8086 cpu. These are the minimum and maximum modes. The minimum mode is used for single processor, minimum chip count applications. The maximum mode allows multiprocessing and attached co-processor systems. All that is required to configure the processor into one or the other of these modes is to tie the MN/MX pin to + 5 volts for minimum mode, or tie the same pin to ground for maximum mode. Depending on the mode chosen, the 8086 will issue all control signals (minimum mode) directly to memory and peripheral devices, or issue status signals to a bus controller (maximum mode) which, in turn, are decoded into the appropriate control signals. The status signals issued by the maximum mode 8086 provide the information necessary for a local bus. This local bus is used to provide the needed information and control to attached co-processors in a multiprocessing environment. Figure 14 illustrates the 8086 configured in the maximum mode. The maximum mode is the mode chosen to be implemented in this library. This is done to allow the use of an attached numeric data processor (NDP), the 8087. To implement the 8087 NDP, current procedure requires the system designer to change the value of the flt flag in the GLOBALS.DAT file from a zero (0) to a one (1). This flag,

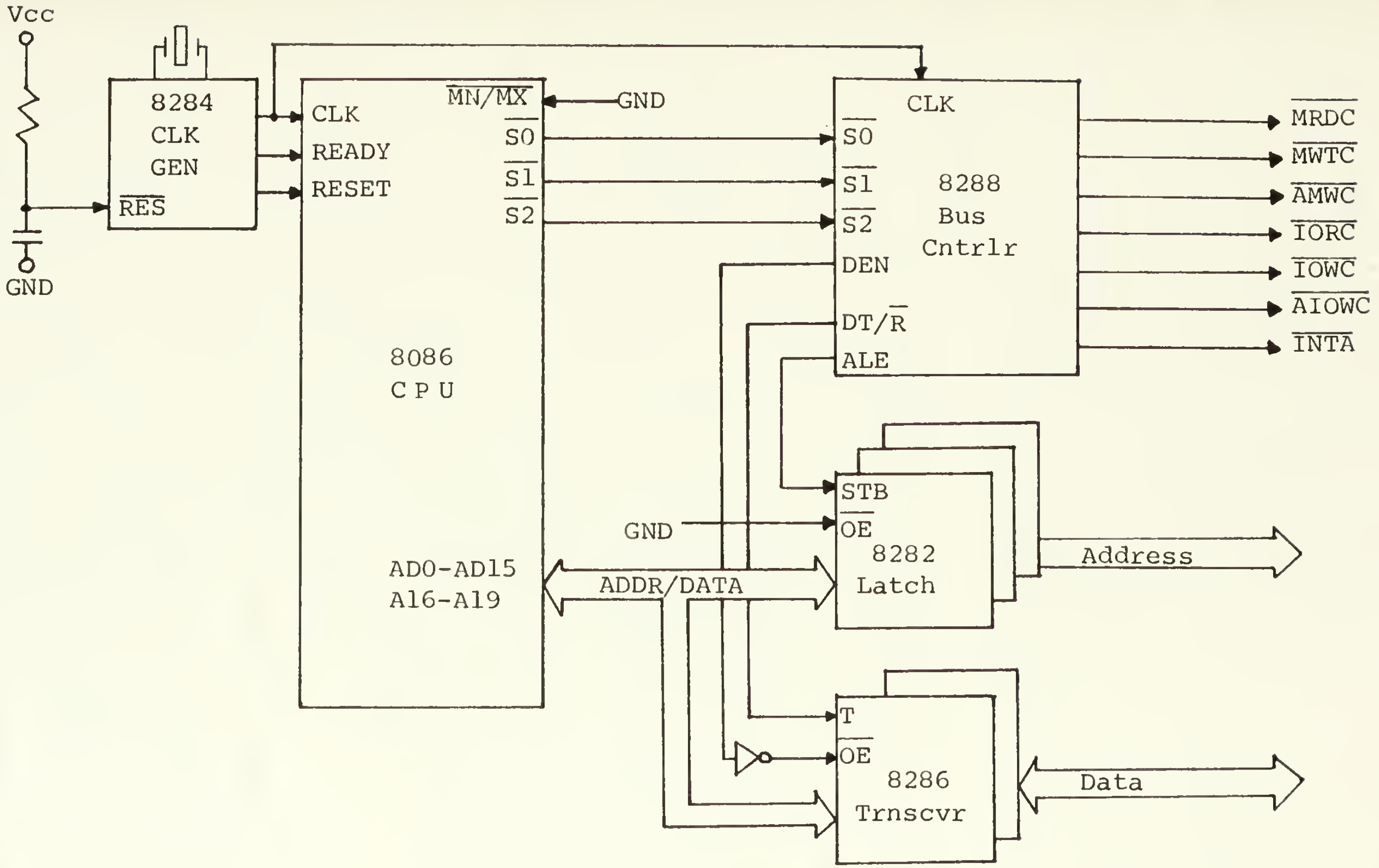


Figure 14. 8086 in Maximum Mode

in turn, calls the 8086 CPU primitive that is configured for exchanging request/grant and queue status signals from/to the 8087.

Timing for the system is provided by an Intel 8284 clock generator, running at 8MHz. This device requires an external 3-times, or 24MHz, crystal for proper operation. In case the option to use the 8087 NDP is chosen, the clock generator must provide a 5MHz frequency using a 15MHz crystal. This is provided for, within the structure of the hardware primitive.

To ensure the proper power levels are available at the data ports of memory, two Intel 8286 octal bus transceivers are used as line drivers. Each transceiver provides 2-way drive for the upper and lower bytes of the 16-bit data word.

Due to the multiplexing of addresses and data on the external pins of the 8086, a requirement exists to latch the address to some other device to allow the data to be made available at some of these same pins. Three Intel 8282 octal latches provide this latching mechanism.

2. Memory and Memory Support Chips

Two types of memory are required to support any system generated. First is some form of ROM that will subsequently receive the instruction code from the software listing. Second is the RAM to be used for data storage and as stack memory as well as RAM for the I/O memory space.

The ROM used in this library are two Intel 27128 16K by 8-bit ultra-violet EPROM's, one for the upper byte, one for the lower byte for every 16K block of memory needed for the system.

Data and stack RAM use sixteen Intel 2167-10 16K by 1-bit static RAM, again, for every 16K block of memory required. This particular memory device is selected to meet the memory access speed requirements of an 8MHz CPU without the necessity for the insertion of wait states. Although the instruction queue in the bus interface unit of the 8086 contains prefetched instructions and data, thereby eliminating most of the requirements to insert wait states, a fast memory has been chosen to meet any immediate memory access requirement (e.g. those imposed by jumps or branches). No other memory options are available, even if the slower 5MHz clock is selected for use with the 8087 NDP.

Input/Output RAM use four Intel 2142 1K by 4-bit static RAM as direct I/O addresses that are available in the 8086.

All ROM and RAM chips are supported by AMD 74LS244 octal three-state buffers as address line drivers integrated into the design to ensure that no single address line has a fan-out of more than 15.

Since 16K blocks of ROM, and data and stack RAM are used, only address lines 1 through 14 (A1 - A14), are needed to select any one byte or word. The full memory is made up

of 32 of these 16K blocks. Therefore, a memory decoding scheme has been implemented to select 1-of-32 16K memory banks. Eight Intel 8205 1-of-8 binary decoders are used in banks of four to select the even and odd byte of the address. The decoders are all driven by address lines A15 through A19 and address zero (A0) for the odd address byte, BHE for the even address byte, with both A0 and BHE active for an aligned 16-bit word (see Figure 15).

3. 8087 Numeric Data Processor

When used, the 8087 NDP extends the register and instruction sets of the 8086 CPU for the purpose of high-speed floating point operations. The hardware implementation of the 8087 NDP is the standard local bus arrangement as shown in Figure 16.

4. Other Hardware Support

Hardware support to meet various possible design requirements is included in the library. Many of the discrete components were taken from Ross' and Pollock's original works and are included in the 8086 library for continuity. A complete list and description of the components are contained in Appendix A.

D. SOFTWARE DESCRIPTION

A listing of the software primitives and their descriptions are contained in Appendix A. Items of particular interest are discussed in more detail in this section.

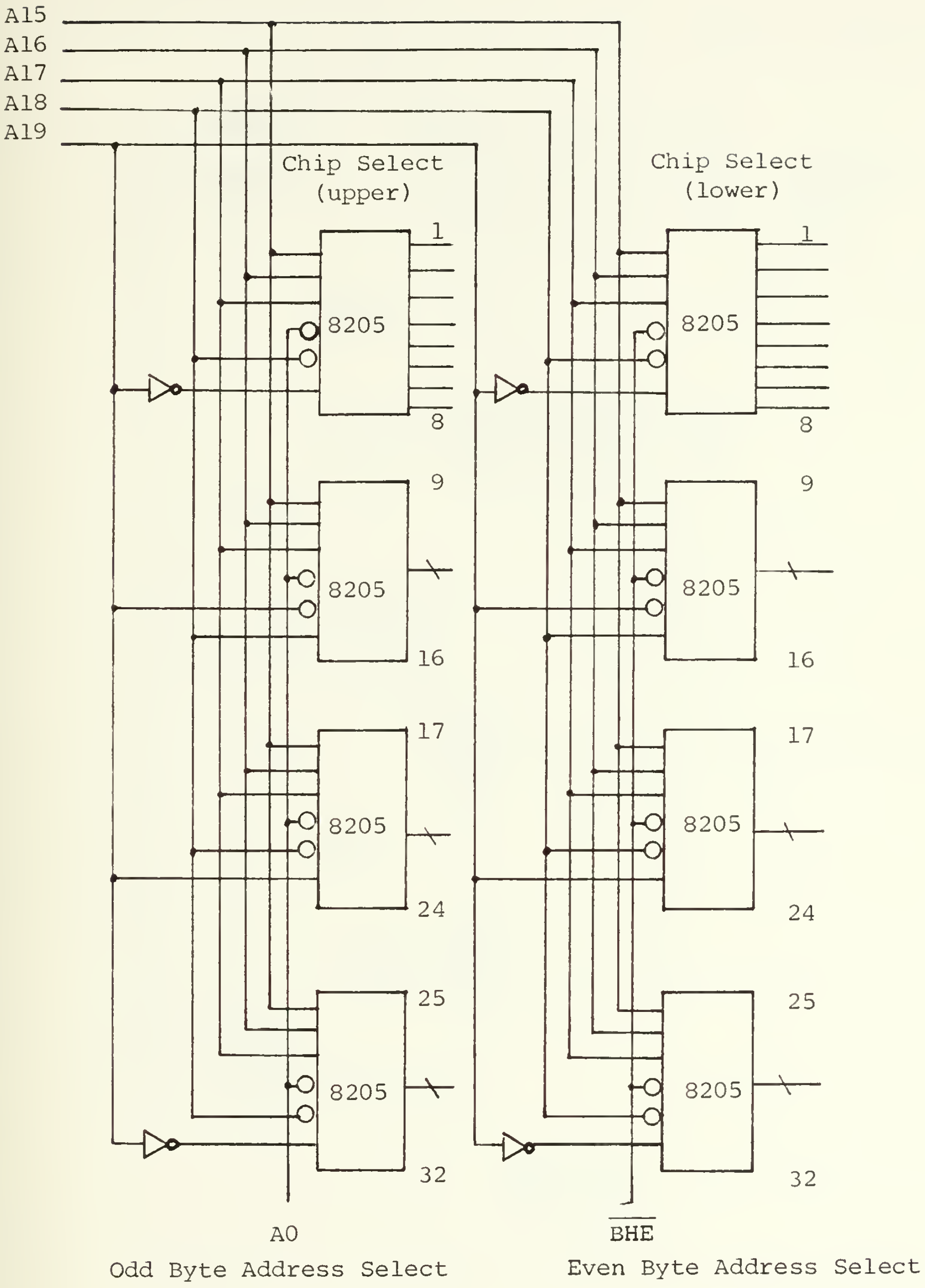


Figure 15. Decoding Memory Banks

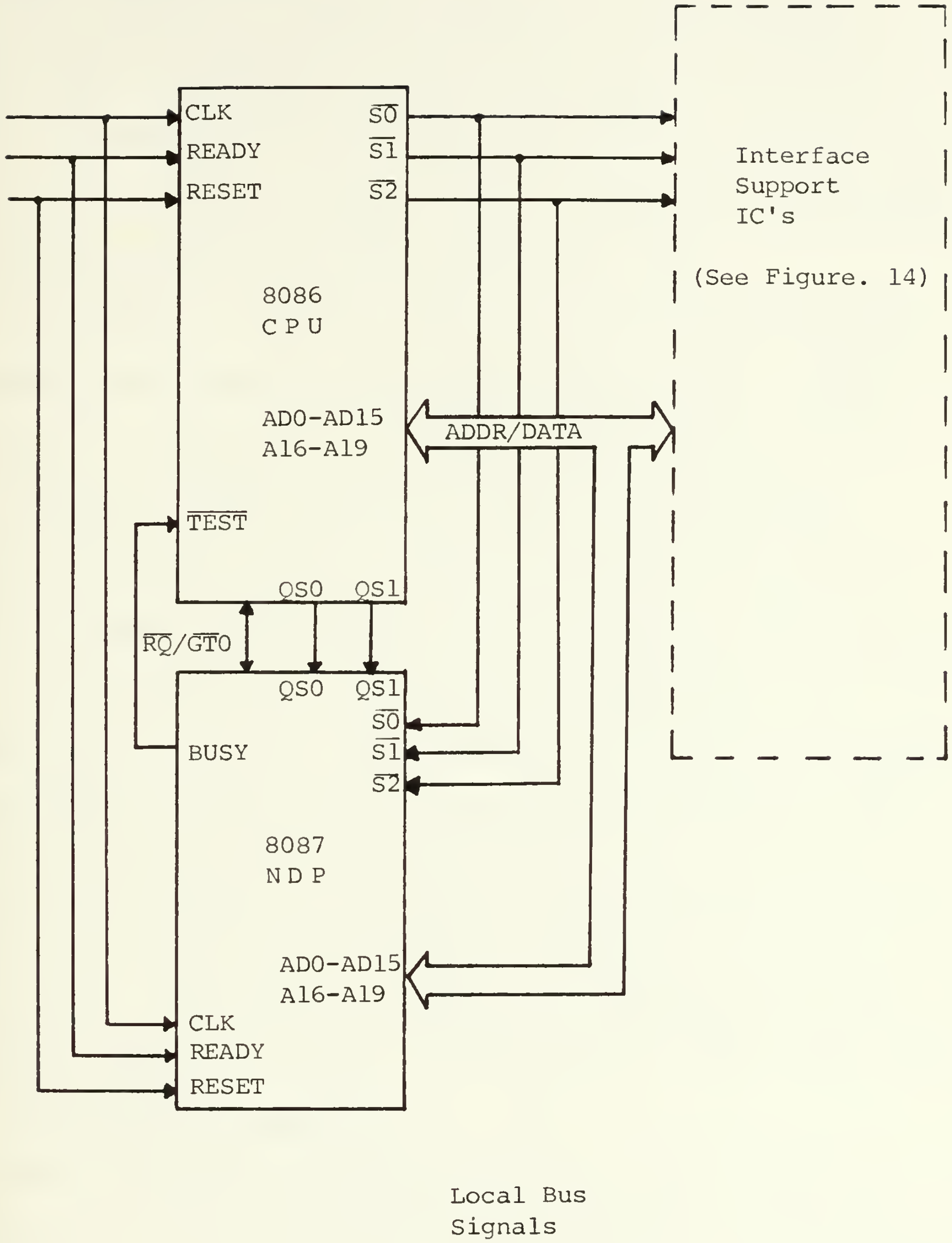


Figure 16. 8087 Numeric Data Processor on Local Bus

1. Monitor Section

The control or monitor section of the software primitive list follows the convention found in the 8080 library. As shown in Figure 17, the monitor consists of a pointer, a table and a section used to test each contingency prior to the execution of its associated task.

A pointer is used to direct the checking of every contingency described in the system's input file. Each contingency is listed in the monitor table in order of actual execution. The pointer's value is the address associated with the current instruction being executed in the monitor table. After each contingency/task check and possible task execution, the value of the pointer is increased to point to the next address in the monitor table. Once the monitor table has been traversed completely, the supervisor is executed, thereby resetting the value of the pointer to the address of the first instruction in the monitor table. This process continues indefinitely until a fatal error occurs or power is turned off to the controller.

2. Calculation Section

Most arithmetic primitives are straight-forward applications of the available 8086 mneumonics. Unlike this libraries predecessor's, where multiply and divide routines contained many lines of code, the 8086 has nmemonics that made these otherwise rigoruos tasks, a simple, single line of code using the various types of the multiply (MUL) and divide (DIV) instructions.

```
;
;       -  monitor section  -
;
@spvsr:   mov    AX,@table    ;initalize table pointer
          mov    @pntr,AX     ;to beginning
@mlop:    mov    BX,@pntr     ;monitor loop
          inc    BX
          inc    BX
          inc    BX
          mov    @pntr,BX
          jmp    BX
;
;       -  data section  -
;
          org    1000         ;ram address pointer
@pntr:    dw     0            ;table entry address pointer
          org    984171       ;rom address pointer
@table:   dw     @pntr        ;table header (define top)
          jmp    @tex1        ;test for contingency ex1
          jmp    @tex2        ;test for contingency ex2
          jmp    @tlstc       ;test for contingency lstc
          jmp    @spvsr       ;go to start of table
;
@tex1:    call   @ex1         ;execute contingency code ex1
          cmp    ex1,1        ;compare contingency result to
                              ;true flag (1)
          jnz    $ + 5        ;if false do not execute stflg
          jmp    @stflg       ;execute task stflg if true
          jmp    @mlop        ;return to monitor
;
@tex2:    call   @ex2         ;execute contingency code ex2
          cmp    ex2,1        ;compare contingency result to
                              ;true flag (1)
          jnz    $ + 5        ;if false do not execute flag2
          jmp    @flag2       ;execute task flag2 if true
          jmp    @mlop        ;return to monitor
          end                 ;software listing complete
```

Figure 17. Example Monitor Section

a. Arithmetic

All simple arithmetic operations, add, subtract, multiply, and divide, are implemented for 8- and 16-bit integer arguments.

(1) Variations in Simple Arithmetic Routines. Along with the simple arithmetic operations, several error checking primitives are available. These are currently used by the system designer as line items in the intermediate primitive list. Once listed in the intermediate file, that specific primitive will be taken from the realization library and used in the software output listing provided that all timing requirements are met. In the future, the designer that uses this system will be queried by the input program whether to incorporate error checking in any or all routines. In all cases, primitives that contain error checking require more memory and take longer to execute than those which have no checking. Many circumstances that arise in the use of a fixed length word or byte as the storage size, provide more than enough bits to cover the range of values of possible arguments. If the designer believes that, for example, an 8-bit add would not cover the possible values of the addends, and the actual result is an important value, as opposed to selecting just the most significant byte for example, then the 16-bit addition without error checking could be used without a penalty of memory used or execution time to implement the addition. An important

point here is that error checking and subsequent correcting, in most cases, substitutes the largest value, either negative or positive depending on whether overflow or underflow occurred, as the value of the result of the arithmetic operation.

Multiplication operations provide several options for the designer's use. These options include 8- by 8-bit or 16- by 16-bit multiplies with options to return an unchecked 8- or 16-bit result, respectively. Also 8- by 8-bit or 16- by 16-bit multiplies, returning a 16- or 32-bit result. Another option is the multiplication of the same size values and returning an unchecked upper or lower byte or word as the result. Appendix A contains a listing of the software primitives as implemented in the realization library of Appendix C.

b. Floating Point

All floating point operations use the Intel short real format. This format is composed of 32-bits with the sign of the mantissa in the most significant bit, followed by an 8-bit signed exponent, and finally followed by a 23 bit normalized mantissa, where the most significant bit of the mantissa is implied to always contain a one. The range available in this format is 8.43E-37 to 3.37E38 [Ref. 8]. To be used as data within a floating point primitive, the library assumes that the 32-bits of the floating point variable passed from the primitive list will be in the

proper double word format, as described above, with the most significant word in the lower storage address and the least significant word in the next higher address. For compatibility with the 8080 library, this data is labeled in byte format, from MSB to LSB, as exponent (exp), high mantissa (hmant), middle mantissa (mmant), and low mantissa (lmant). This method of storing a floating point variable is also compatible for use with the 8087 NDP. Figure 18 illustrates the composition of a floating point variable using this format.

(1) Use of the 8087 Numberic Data Processor. Current implementation of floating point arithmetic makes use of the 8087 NDP only. No stand alone software for floating point manipulations are included in the library. However, provisions have been made to incorporate software-only floating point operation by writing primitives to pack and unpack floating point variables into and out of the short real format for ease of manipulation within the arithmetic routines. The unpack primitive separates the mantissa sign bit, the exponent, and the mantissa into individual bytes, words or double words and is then stored into a designated scratchpad area in RAM. Upon completion of a floating point operation, the values are packed into the proper format prior to being stored as the result. The current implementation must pack and unpack variables each time they are

Labels given to each byte of a floating point variable or constant (provides compatibility between 8080 and 8086 libraries)

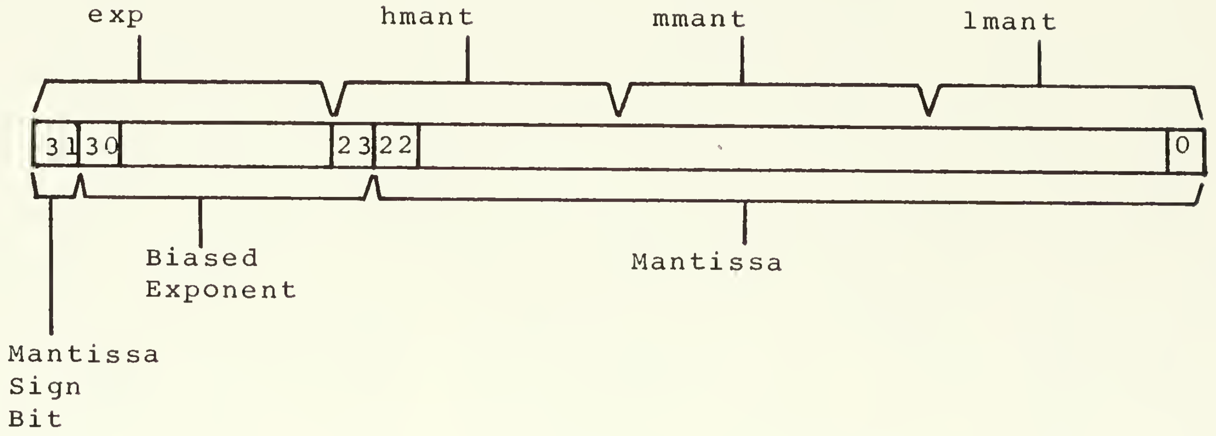
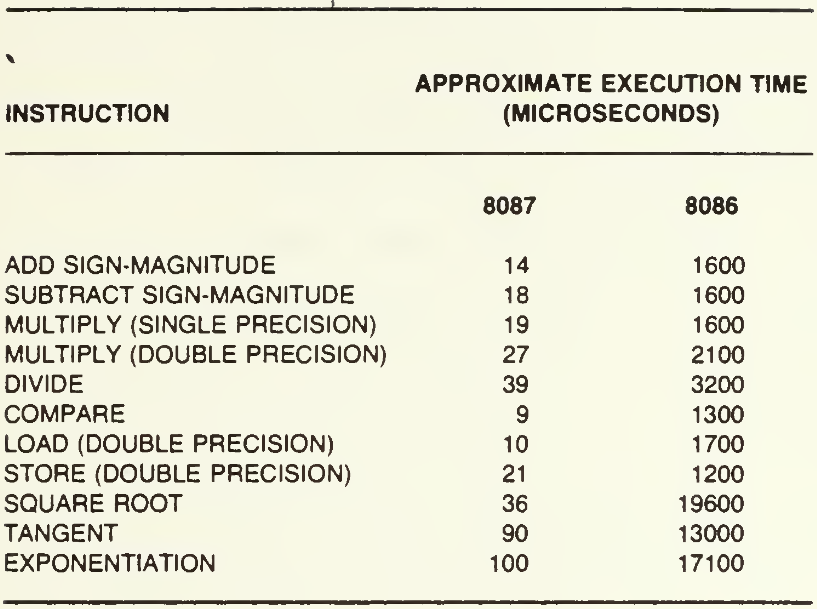


Figure 18. Floating Point Variable Format

used, since there is no method to keep track of variables that have been previously used during the execution of a systems design.

The choice for using the 8087 NDP for all floating point operations, as shown in Figure 19 [Ref. 9], is speed. Without the 8087, a single precision floating point addition would require about 6.2 milliseconds to complete in an optimized environment. This takes into account two single precision loads, one floating point addition, and one single precision store. This is in comparison to 53 microseconds that an 8087 NDP would require to accomplish the same operation. This is a better than a 100 times improvement over the 8086 software floating point addition. In many cases, for a real-time controller design, 6.2 milliseconds will not satisfy speed requirements for multiple floating point computations.

Prior to any 8087 floating point operation, the NDP must be initialized by the loading of a control word. The control word provides programming options for allowing or disallowing the recognition of interrupts, precision specification (data type selection), rounding control, infinity control and exception handling options. The control word used in the initialization of the 8087 NDP in the library are selected as follows:



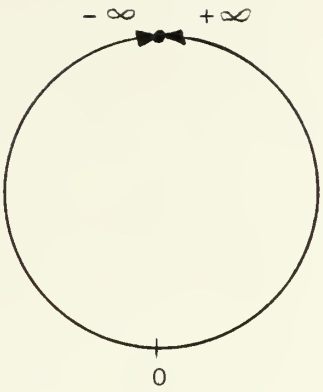
INSTRUCTION	APPROXIMATE EXECUTION TIME (MICROSECONDS)	
	8087	8086
ADD SIGN-MAGNITUDE	14	1600
SUBTRACT SIGN-MAGNITUDE	18	1600
MULTIPLY (SINGLE PRECISION)	19	1600
MULTIPLY (DOUBLE PRECISION)	27	2100
DIVIDE	39	3200
COMPARE	9	1300
LOAD (DOUBLE PRECISION)	10	1700
STORE (DOUBLE PRECISION)	21	1200
SQUARE ROOT	36	19600
TANGENT	90	13000
EXPONENTIATION	100	17100

Figure 19. Speed Comparison Between 8087 NDP and 8086

1) interrupts disabled
2) short real data type (24 bits of precision)
3) round to nearest value or even
4) projective infinity control

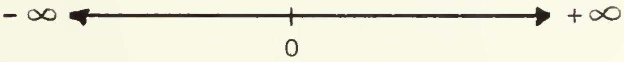
Items 1 through 3 are self explanatory. Item 4, projective infinity control, provides the control necessary to close the system of numbers in the 8087. Although affine closure provides more information than projective, there is a greater chance for misinformation by using the affine closure mode if, for example, the reciprocal of plus or minus zero is computed. Figure 20 shows that the result of this computation would give two different results for the same values in the operation. The projective mode never returns misinformation and is therefore the suggested mode for use when the values of an operation are not known in advance. [Ref. 10]

Due to the serial nature of the processing environment of CSDL and the 8086/8087 structure within the environment, optimal use of the processing group is sacrificed for the stand-alone speed capabilities of the 8087. As shown in Figure 21a [Ref. 11], the 8086 in a co-processing environment with the 8087 is afforded the ability to continue fetching and executing its instruction stream while the 8087 is performing a more involved floating point operation. Once the 8087 has completed its task, it interrupts the 8086 just long enough to store the result of its previous computation. In this library implementation,



Projective Closure

Projective closure provides a completely closed set of numbers, thereby never providing misinformation.



Affine Closure

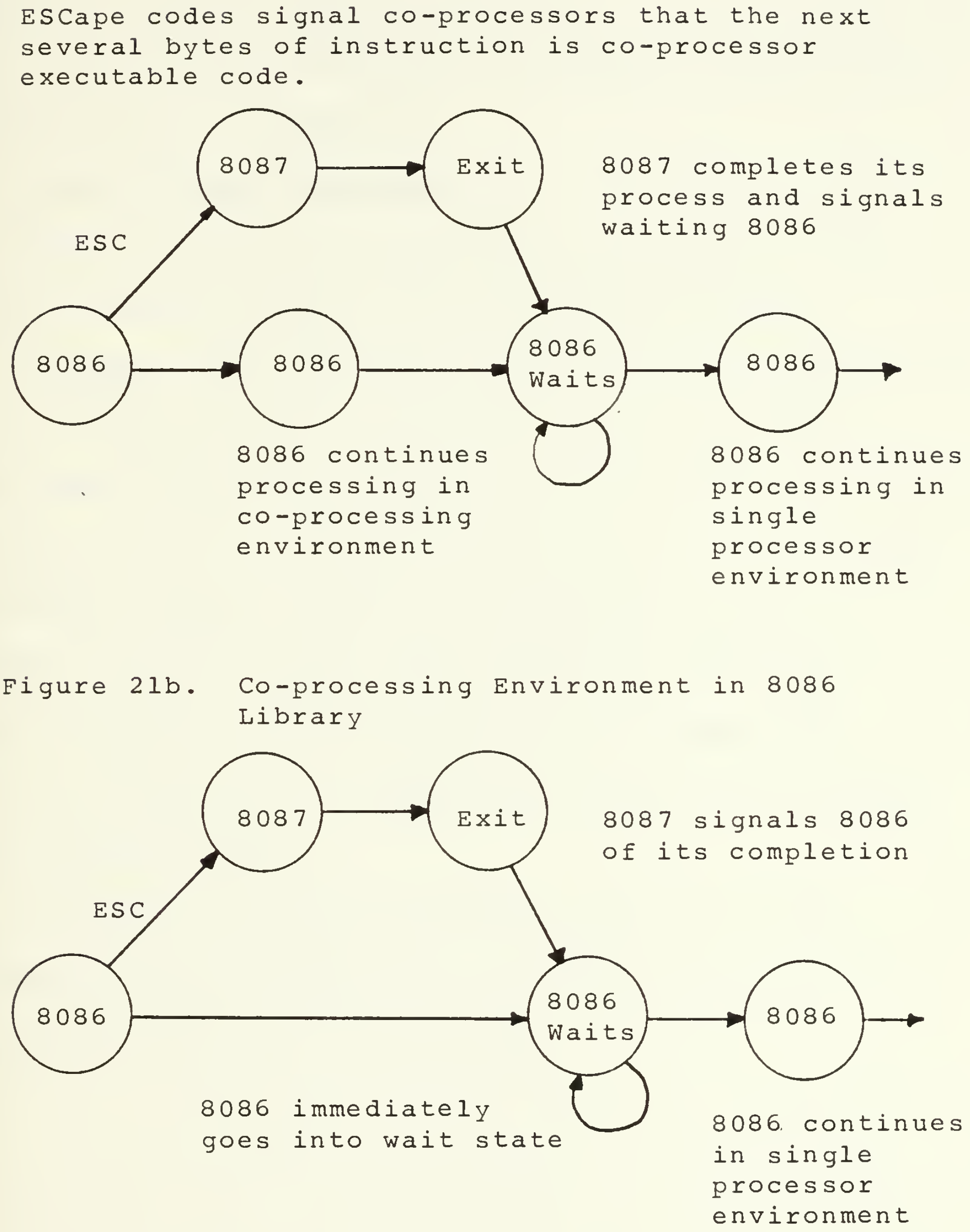
More information is available with Affine closure but is not recommended when the values of variables are not always known

Figure 20. Projective versus Affine Closure

as the environment of system design forces serial calculations, the 8086 is immediately dependent on the outcome of the 8087's current calculation. This requires the 8086 to enter an unproductive idle state as shown in Figure 21b. The overall calculation speed of this system is increased, however, at the expense of adding 8087 NDP hardware for floating point operations. A tradeoff in overall system speed must be made when the 8087 NDP is used, since the 8086 CPU must be slowed to a 5MHz rate with the addition of the 8087 hardware.

(2) Emulation of the 8087 Numeric Data Processor. If speed is unimportant in a particular realization which contains floating point operations, and the decision not to use the 8087 NDP is made, an alternative to creating complete software-only primitives for floating point calculations is the use of the 8087 emulation package that provides the software equivalent of the 8087. With this package installed, no difference exists between a routine that will run on the 8087 or 8086 emulation of the NDP. The decision to use the 8087 hardware is made at link time, with no further re-assembly required to produce the proper 8086 code [Ref. 12]. All that is required, in this case, would be to substitute the values for execution cycles in the software primitives of the library to ensure an accurate timing analysis is performed prior to the generation of the system design listings.

Figure 21a. Normal Co-processing Environment



ESCape codes signal co-processors that the next several bytes of instruction is co-processor executable code.
8087
Exit
8087 completes its process and signals waiting 8086
ESC
8086
8086
8086 Waits
8086
8086 continues processing in co-processing environment
8086 continues processing in single processor environment

Figure 21b. Co-processing Environment in 8086 Library

8087
Exit
8087 signals 8086 of its completion
ESC
8086
8086 Waits
8086
8086 immediately goes into wait state
8086 continues in single processor environment

Figure 21. Use of ESC Codes in a Co-processing Environment

3. Logical Functions

Included in this library is a complete set of software primitives to perform logical operations. These include 8- and 16-bit and, or, not, xor, eq, lt, le, gt, ge, and ne. All of these primitives are compatible with those in the 8080 and Z-80 libraries.

4. Other Software Functions

Among the remaining primitives contained in this library are program control and input/output control.

Program control includes those primitives required to establish storage areas for constants, set up variable storage areas, define line labels to be used for conditional jumps, jump on true or false, as well as primitives to set up for-loops, while-do loops, and establish and call procedures. A fixed-length delay primitive allows the designer to specify a delay, to wait for an input for example, in increments of 5 microseconds for both the 8MHz and 5MHz design versions.

Input/Output control provides for 8- or 16-bit input/output bytes or words. The software primitives are taken from the original 8080 library.

V. TESTING OF SOFTWARE PRIMITIVES

A. PRIMITIVE TESTING VEHICLE

Software primitives for this library were developed and tested on the Naval Postgraduate School Mathematics Department's North Star Computer, Inc., "Horizon" computer system, augmented with an Octagon Computer System's "Octagon Board 8/16" plug in processor board. This S-100 board gives to the normally Zilog Z-80 based "Horizon" an 8086/8087 processor capability. The 8087 Numeric Data Processor is not currently installed on the 8/16 board. Those primitives that make use of the 8087 NDP are, therefore, not operationally tested. Primitives that use 8087 code are taken in part from examples given in the book, 8087 by Richard Statz [Ref. 13]. Although these 8087 primitives may not function properly, if implemented, they are included as a starting point for future investigation.

In conjunction with the use of the "Horizon" computer, the testing environment operated under CP/M-86. All programs writtenfor verification purposes to be used as primitives in this library were assembled using ASM-86. It should be noted that other assemblers may have directives that do not equate to those directives used in this library, and any assembly errors may be due to the incompatability of these directives.

B. TESTING OF CONTROL LOGIC

All primitives that contain 'if' lines, and therefore have conditional branches, have been tested for all possible branches. For example, the addition of an 8087 NDP requires that the clock period attribute be changed from 0.125 microseconds to 0.2 microseconds, the crystal used with the 8284 clock generator be changed from 24MHz to 15MHz, and all the associated changes be made to accommodate the slower overall system speed. In this example, the FLT global variable is used as a flag to signify the addition of the 8087.

VI. RECOMMENDATIONS AND CONCLUSIONS

A. RECOMMENDATIONS

1. Universal High-Level Language Realization Library

A large scale reproduction of basic software primitives required to create another realization library is far too labor intensive to be performed for each new microprocessor family. The following paragraphs explain how this section of library creation may be avoided.

A single high-level language library created by developing software primitives, would provide instant compatibility between different microprocessor families. Once a high-level language library is written, the use of a cross compiler/assembler could be used to construct the actual library used by the timing analyzer and formatter. Cross compilers are currently available to convert C language programs to any number of target microprocessor machine code programs [Ref. 14]. The hardware portion of the new library would still be constructed component by component. After the cross compiler produces an assembly language/machine code file for timing analysis, the appropriate hardware primitives are appended to the assembly language file. Another volume of timing and size primitives, constructed to match the microprocessor, would provide the number of cycles and bytes required to realize a given

design. This timing/size companion volume would simply be a rewritten version of the new microprocessor's index of mnemonics, usually provided in the processor programming guide. An example of this process is shown in Figure 22.

2. Hardware Wiring Diagrams

The current method to implement hardware components is left to the realization library author. Although pin-out descriptions are simple, several areas in the hardware text section is free-form. With the potential to diagram the hardware realization to a high resolution graphics terminal, a more structured format in the hardware section is suggested.

B. CONCLUSIONS

The integration of the Intel 8086 microprocessor realization library into the overall design capabilities of CSDL offers comparable primitives to that of the original works by Ross and Pollock. A universal globals file (globals.dat), as constructed in this paper, must be maintained to allow the design criteria section of CSDL to migrate from microprocessor to microprocessor library to generate the desired system by meeting the design specifications. This option was not maintained by Smith in his prototyping scheme for the Z-80 realization library, and therefore, his specialized globals are not included in the universal globals file. Libraries constructed for use by this design system should continue to be produced in order

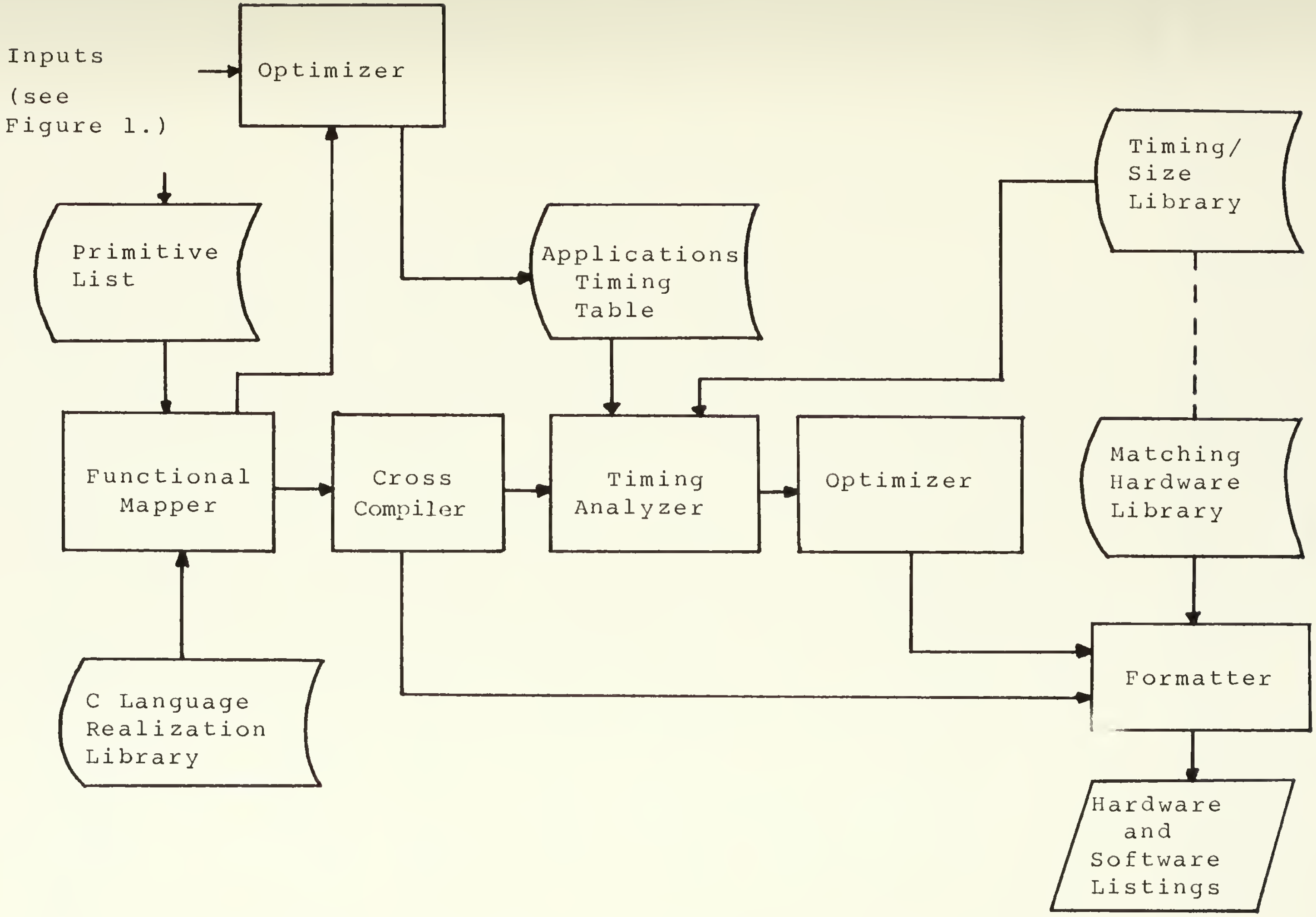


Figure 22. Universal High Level Language Software Library Implementation

to provide a wide range of processor families from which to choose. With the eventual integration of a "user friendly" input method, CSDL will provide an advanced environment in which to design and implement real-time microprocessor-based controllers.

APPENDIX A

DEFINITIONS OF PRIMITIVES

List of hardware and software primitive functions. The primitive definitions preceded by an (*) are from the 8080 library that are also used in the 8086 library.

h.adc. * Defines an 8-bit analog to digital converter

h.adc2 * Defines a 8-bit processor-controlled analog to digital converter.

H.bufframp * Defines a buffer input amplifier.

h.capac-cer * Defines any value ceramic capacitor.

h.clckctrs * Defines a 16-bit TTL counter for a free running clock.

h.conn-al * Defines dip socket used as an analog connector.

h.conn26sep * Defines a 26 pin flat cable connector.

h.dac * Defines a 8-bit digital to analog converter.

h.diode-sw * Defines a silicon switching diode.

h.diode-znr * Defines a zener diode.

h.drvr Primitive to implement address line drivers as more 16K banks of memory are added to the hardware realization.

h.eprom Defines the pin-out of an Intel 27128 128k ultra-violet erasable programmable read-only memory.

h.ffjk * Defines a TTL flip-flop.

h.follower * Defines a voltage follower.

h.herr Performs a simple check to determine if ROM space overflows into RAM space and prints a warning message if an overflow occurs.

h.invert * Defines a TTL inverter.

h.ioram Defines the pin-out of an Intel 2142 4k static RAM for input/output memory.

h.issuecond * Defines an 8- or 16-bit condition-type output hardware.

h.issuesepcd * Defines a single bit output.

h.issuevent * Defines event-type output hardware.

h.memory Calculates the required read-only and read-write(ram) memory chips required to implement the current design. It also assigns the chip select line numbers generated by the address decoder hardware primitive that is contained in the processing unit primitive, h.processor.

h.nand2 * Defines a 2 input discrete nand gate.

h.nand3 * Defines a 3 input discrete nand gate.

h.nand4 * Defines a 4 input discrete nand gate.

h.nand8 * Defines an 8 input discrete nand gate.

h.ndp Defines the pin-out of the Intel 8087 Numeric Data-Processor for high speed floating-point calculations.

h.nprocessor Defines the pin-out of the Intel 8086-2 8MHz microprocessor for use with an Intel 8087 Numeric Data Processor.

h.oneshotl * Defines a TTL edge triggered, retriggerable one shot.

h.opamp * Defines operational amplifier hardware.

h.opisol2 * Defines a slow photo darlington optical isolator.

h.optoisol * Defines optically isolated logic device.

h.oscxtal * Defines a modular crystal oscillator.

h.peripdrivr * Defines peripheral driver hardware.

h.processor Defines the pin-out of the Intel 8086-2 8MHz microprocessor for use without any attached coprocessor.

h.ram Defines the pin-out of the Intel 2167 16k static read/write(ram) memory chip.

h.relaydpdt * Defines a double-pole double-throw relay.

h.resmfqtrwt * Defines 1/4 watt metal film 1% resistor.

h.rpack-18b * Defines 8 resistor, 180 ohm each, resistor package in a 16 pin dip.

h.rs232conn * Defines rs-232c input/output connection.

h.rs232rx * Defines an rs-232c receiver.

h.rs232tx * Defines an rs-232c transmitter.

h.sensecond * Defines 8- or 16-bit condition-type input hardware.

h.sensepcond * Primitive to define a single bit input device.

h.sensevent * Primitive to define an event-type input hardware.

h.striginvrt * Defines a TTL schmidt trigger inverter.

h.support Defines the support IC's required for use with both variations of the 8086 processor. This includes the following chips:

1. Intel 8284 Clock Generator and driver.
2. Intel 8286 Bus Transceiver.
3. Intel 8288 Bus Controller.
4. Intel 8282 Octal Latch.
5. Intel 8205 1-of-8 Decoder.

h.trimpot * Defines a variable resistor of (r) ohms with a 1/2 watt trimpot.

h.uart * Primitive to define a UART.

s.8087cw Routine to load the 8087 control word prior to the first 8087 NDP operation. The control word specifies:

1. Intel short form for floating point data representation.

2. Rounding to nearest or even value.
3. Projective infinity control.

s.add Primitive to add two 8- or 16-bit integer numbers.

s.addck Primitive to add two 8- or 16- bit integer numbers with overflow and underflow checks. If an overflow error exists the largest positive number is stored as the result. On an underflow condition, the largest negative number is then stored as the result.

s.anain * Primitive to define a processor-controlled analog input.

s.analogin * Primitive to define an analog input condition.

s.anaout * Primitive to define an analog output channel.

s.and Primitive to perform a logical 'and' of two 8- or 16- bit numbers.

s.assign * Primitive to assign an 8- or 16-bit value of one variable to another variable.

s.assigncons * Primitive to assign an 8- or 16-bit constant to a variable.

s.clock * Primitive to define a free running time clock.

s.cons Primitive to define a data constant to be used at program assembly time.

s.div Divide routine for 8- or 16-bit unsigned division.

s.end Defines the end of the output assembly language listing.

s.eq Primitive to determine if two 8- to 16- bit integers are equal.

s.eventmark * Primitive to establish interrupt handler for event.

s.every Primitive to define dummy function for every-period statement.

s.exitproc Closes a procedure and resets the associated contingency.

s.fadd Routine to perform floating point addition.

s.fassign Routine to assign the value of one floating point variable to another floating point variable.

s.fcons Routine to define storage for a floating point constant in short real form.

s.fdiv Routine to perform non-numeric data processor floating point divide.

s.fix Routine to convert a floating point value to an integer value.

s.fixedwait Routine to delay a fixed period of time in 5 microsecond increments for an 8MHz clock.

s.fixedwait5 Routine to delay a fixed period of time in 5 microsecond increments for a 5MHz clock.

s.float Routine to convert an integer value into a short real format floating point value.

s.fmul Routine to perform non-numeric data processor floating point multiply.

s.forcons Defines the start of a constant bounds loop.

s.forend Defines the end of a constant bounds loop.

s.fpack Routine to pack an unpacked floating point result of a non-numeric data processor floating point operation. This puts the result of a non-numeric data processor floating point operation back into memory in short real form. (See s.funpack).

s.fsub Routine to perform a non-numeric data processor floating point subtraction.

s.funpack — Routine to unpack a floating point argument prior to a non-numeric data processor floating point operation. The argument is stored in short real form prior to unpacking. (See s.fpack).

s.fvarset — Primitive to establish a scratch-pad in memory to handle non-numeric data processor floating point operations.

s.ge — Determines if argument1 is greater than or equal to argument2 for 8- or 16-bit numbers.

s.gt — Determines if argument1 is greater than argument2 for 8- to 16-bit numbers.

s.heading — Primitive to define assembly language software heading.

s.imul16 — Primitive to multiply two 16-bit signed integer numbers without overflow checking and returns a 16-bit result.

s.imul8 — Primitive to multiply two 8-bit signed integer numbers without overflow checking and returns an 8-bit result.

s.imulex16 — Primitive to multiply two 16-bit signed integer numbers and returning a 32-bit result.

s.imulex8 — Primitive to multiply two 8-bit signed integer numbers and returning a 16-bit result.

s.imulom16 — Primitive to multiply two 16-bit signed integer numbers with overflow and underflow checks. On overflow, the maximum positive 16-bit signed number is returned as the result. On underflow, the maximum negative 16-bit signed number is returned as the result.

s.imulom8 — Primitive to multiply two 8-bit signed integer numbers with overflow and underflow checks. On overflow, the maximum positive 8-bit signed number is returned as the result. On underflow, the maximum negative 8-bit signed number is returned as the result.

s.imulul6 Primitive to multiply two 16-bit signed integer numbers returning the upper 16 bits as the result.

s.imulu8 Primitive to multiply two 8-bit signed integer numbers returning the upper 8 bits as the result.

s.in Routine to set the timed block flag.

s.issuecond Routine to send condition-type output.

s.issueopcnd * Primitive to define an optically isolated digital output.

s.issueevent * Primitive to send event-type output.

s.jmpf Routine to branch on false condition.

s.jmpt Routine to branch on true condition.

s.le Determines if argument1 is less than or equal to argument2 for 8- or 16-bit numbers.

s.loc Routine to define a label.

s.lt Determines if argument1 is less than argument2 for 8- or 16-bit numbers.

s.main Primitive for software initialization. Sets event, port, rom and ram pointers and calls hardware processor primitive and software heading primitive.

s.monitor Primitive to define the p2 monitor as controller supervisor.

s.mul Primitive to multiply two 8- or 16-bit unsigned integer numbers without error checking. Returns 8- or 16-bit results.

s.ne Determines if argument1 is not equal to argument2 for 8- or 16-bit numbers.

s.nfadd Routine to perform numeric data processor implemented floating point addition.

s.nfdiv Routine to perform numeric data processor implemented floating point divide.

s.nfmul Routine to perform numeric data processor implemented floating point multiplication.

s.nfsub Routine to perform numeric data processor implemented floating point subtraction.

s.ni Primitive to clear timed block flag.

s.nopck Routine to handle overflow and underflow conditions resulting from 8087 NDP arithmetic operations.

s.not Performs 8- or 16-bit logical 'not'.

s.or Performs 8- or 16-bit logical 'or'.

s.perform Routine to invoke a procedure.

s.proc Used to define a procedure entry point.

s.relayout * Primitive to define a relay output.

s.restans * Primitive to define a resistance transducer.

s.senscontac * Primitive to sense a contact closure.

s.sensecond Primitive to detect condition-type input.

s.sensevent * Primitive to detect an event-type input.

s.senshotct * Primitive to detect a hot contact.

s.sensopcond * Primitive to define an optically isolated condition input.

s.sensopevt * Primitive to define an optically isolated event input.

s.start Dummy start primitive.

s.sub Primitive to subtract two 8- to 16-bit numbers.

s.tabaccp2 Subroutine to add routine access for contingency/task pair.

s.tabend Defines the end of the monitor table.

s.tabent Primitive to add an entry to the monitor table.

s.temp	* Primitive to define temperature measurement channel.
s.var	Routine to define storage for 8- or 16-bit integer variable.
s.whend	Primitive marking the end of a while-do statement.
s.whilecon	Primitive to generate a while-do condition section head.
s.xor	Perform 8- or 16-bit logical 'exclusive or'.

APPENDIX B

LIST OF GLOBAL VARIABLES

The following global variables list is used by the realization library given in Appendix C. This list incorporates all the global variables used in the 8080 and 8086 libraries, allowing compatibility between the two and thus not requiring any user intervention in system choice of microprocessors selected during system generation. The value in parenthesis following each variable is its initial value.

CN (0) Counter for capacitor reference designator.

CNT (0) Counter for use during calculation in the fixed wait primitive.

CRN (0) Counter for diode reference designator.

CWORD (0) Flag for loading 8087 NDP control word (8086 Library only).

DIV (0) Flag indicating inclusion of division subroutines.

EVADDR (0) Event address.

EVPNT (0) Event pointer.

FLT (0) Flag indicating the use of floating point processor.

ICN (0) Counter for integrated circuit reference designator.

INE (1) Element counter for multi-element invertor.

INN (0) Reference designator for multi-element invertor.

INPORT (0) Pointer to next available input port.

ISCE (1) Element counter for bit slice output port.

ISCN (0) Reference designator for bit sliced output port IC.

ISCP (0) Pointer to bit sliced output port.

ISNE (1) Element counter for multi-element Schmitt-trigger invertor.

ISNN (0) Reference designator for multi-element Schmitt-trigger invertor.

JAASE (1) Element counter for bit sliced I/O connector.

JAASN (0) Reference designator for bit sliced I/O connector.

JKE (1) Element counter for multi-element J/K flip-flop.

JKN (0) Reference designator for multi-element J/K flip-flop.

JN (1) Counter for connector reference designator.

KN (1) Counter for relay reference designator.

LATFLG (0) Flag for pulse event inputs requiring latching.

MEM (0) Pointer used in ROM/RAM partition.

MPRTA (0) Pointer to data I/O port of hardware floating point processor.

MPRTB (0) Pointer to control I/O port of hardware floating point processor.

MUL (0) Flag indicating inclusion of multiplication subroutines.

NBE (1) Element counter multi-element 2 input nand gate.

NBN (0) Reference designator for multi-element 2 input nand gate.

NCE (1) Element counter for multi-element 3 input nand gate.

NCN (0) Reference designator for multi-element 3 input nand gate.

NDE (1) Element counter for multi-element 4 input nand gate.

NDN	(0)	Reference designator for multi-element 4 input nand gate.
ODT	(0)	Flag for use of debugging package (8080 Library only).
OUTPRT	(-1)	Pointer to next available output port.
PDE	(1)	Element counter for multi-element peripheral driver.
PDN	(0)	Reference designator for multi-element peripheral driver.
RABE	(1)	Element counter for multi-element resistor pack.
RAMPTR	(0)	Pointer to next address in RAM.
RN	(1)	Counter for resistor reference designator.
ROMPTR	(0)	Pointer to next address in ROM.
RPN	(1)	Counter for resistor pack reference designator.
RSDE	(1)	Element counter for multi-element RS-232C driver.
RSIC	(0)	Reference designator for multi-element RS-232C driver.
RSRIC	(0)	Reference designator for multi-element RS-232C receiver.
RSRE	(1)	Element counter for multi-element RS-232C receiver.
SCRTCH	(0)	Scratch register.
SSCHE	(0)	Element counter for bit sliced input port.
SSCP	(0)	Pointer to bit sliced input port.
TMBLCK	(0)	Flag for timed block.
TOTEVT	(0)	Total count of event type inputs.

APPENDIX C

8086 CPU REALIZATION LIBRARY

```
v0000intel 8086 cpu : clkper=0.125 : memdly=0.125 : moncst=10:
v3074h.adc        (in,out,h,1:0,8,25,100,0,-100:0,400,1,93,89,3074,3168)
v4175h.adc2       (in,out,h,1:0,8:0,800,1,50,7,4175,4226)
v3276h.bufframp   (in,ret,out,g,b::0,150,1,22,5,3276,3313)
v4091h.capac-cer  (in,out,val:10,9900000:0,0,0,10,0,4091,4102)
v3404h.clckctrs   (::0,1000,1,24,0,3404,3503)
v4130h.conn-al    (in,ret,shld,name::0,0,0,11,0,4130,4142)
v2920h.conn26sep  (s,r,sh,t::0,0,0,4,0,2920,3005)
v3169h.dac        (in,out,h,1:0,8:0,350,1,68,0,3169,3238)
v4103h.diode-sw   (sigpos,signeg::0,0,0,10,0,4103,4114)
v4115h.diode-znr  (sigpos,signeg,v:1,200:0,0,0,13,0,4115,4129)
v0927h.drvr       (::0,135,2,15,0,927,954)
v0603h.eprom      (cnt::0,150,2,15,0,603,632)
v3784h.ffjk       (j,k,q,nq,s,r,ck::0,0,0,4,0,3784,3835)
v3239h.follower   (in,out::0,150,1,16,0,3239,3256)
v1046h.herr       (::0,0,0,0,0,1046,1076)
v3504h.invert     (in,out::0,0,0,4,0,3504,3562)
v0633h.ioram      (::0,200,4,16,0,633,692)
v2493h.issuecond  (signam,:0,8,0,128: 2,500,1,17,0,2493,2519)
v2520h.issuecond  (signam:9,16,0,32000:0,0,0,11,10,2520,2541)
v2668h.issuesepcd(s:0,8,0,128:2,0,0,4,0,2668,2746)
v2486h.issuevent  (signam,outprt:0,8,0,128: 2,500,1,0,5,2486,2492)
v1010h.memory     (:0,1048576:0,0,0,9,7,1010,1045)
v3622h.nand2      (a,b,o::0,0,0,4,0,3622,3675)
v3676h.nand3      (a,b,c,o::0,0,0,4,0,3676,3723)
v3724h.nand4      (a,b,c,d,o::0,0,0,4,0,3724,3762)
v3763h.nand8      (a,b,c,d,e,f,g,h,o::0,60,1,19,0,3763,3783)
v0570h.ndp        (::8,3000,1,31,0,570,602)
v0223h.nprocessor(::5,2500,1,32,33,223,258)
v3378h.oneshot1   (a,b,c,d,name,time:20,20:0,125,1,0,23,3378,3403)
v3257h.opamp      (p,n,o,t,u::0,85,1,17,0,3257,3275)
v2618h.opisol2    (in,ret,outcol,outem::0,110,1,15,0,2618,2642)
v2597h.optoisol   (signam,sigret,sigout::0,95,1,19,4,2597,2617)
v3365h.oscxtal    (name,freq:1,200:0,300,1,11,0,3365,3377)
v2747h.peripdrivr(a,b,o::0,0,0,4,0,2747,2779)
v0188h.processor  (::8,2500,1,32,33,188,222)
v0693h.ram        (cnt::0,720,18,15,0,693,926)
v2780h.relaydpdt  (s,v,t:0,2:0,2000,0,16,15,2780,2797)
v3999h.resmfqtrwt(sigin,sigout,ohms:1,2000000:0,0,0,9,0,3999,4009)
v4010h.rpack-18b  (in,out::0,0,0,4,0,4010,4078)
v3836h.rs232conn  (in,sg,out,pg::0,0,0,11,0,3836,3848)
v3894h.rs232rx    (in,out::0,0,0,9,0,3894,3948)
v3849h.rs232tx    (in,out::0,0,0,4,0,3849,3893)
v0955h.sensecond  (signam,inport:0,8,0,1024:2,500,1,18,0,955,974)
v0975h.sensecond  (signam,inport:0,16,0,1024:2,1000,2,18,0,975,1009)
v2843h.sensepcond(s:0,8,0,128:2,0,0,4,0,2843,2919)
v2456h.sensevent  (signam,evpnt : 1,8,0,128: 2,500,1,22,21,2456,2479)
v3563h.striginvrt(in,out::0,0,0,4,0,3563,3621)
v0259h.support    (::8,6610,15,33,0,259,569)
v4079h.trimpot    (in,out,w,r:50,2000000:0,0,0,10,0,4079,4090)
v3949h.uart       (si,so,rc,tc::0,500,1,3,0,3949,3998)
v1862s.8087cw     (::3,20,3,11,0,1862,1874)
```

```
v1586s.add       (rslt,arg1,arg2:0,8,0,8,0,8:9,41,12,10,0,1586,1597)
v1640s.add       (rslt,arg1,arg2:0,16,0,16,0,16:12,44,12,10,0,1640,1651)
v1598s.addck     (rslt,arg1,arg2:0,8,0,8,0,8:21,96,25,19,0,1598,1618)
v1619s.addck     (rslt,arg1,arg2:0,16,0,16,0,16:21,96,25,19,0,1619,1639)
v4143s.anain     (sig,h,1,b:0,8,50,-25,1,100:0,0,0,0,11,4143,4174)
v3006s.analogin  (sig,h,1,b:0,8,100,-100,1,100:0,0,0,0,10,3006,3050)
v3051s.anaout    (sigout,h,1:0,8,25,100,0,-100:0,0,0,0,10,3051,3073)
v2152s.and       (rslt,arg1,arg2:0,8,0,8,0,8:10,41,11,10,0,2152,2163)
v2164s.and       (rslt,arg1,arg2:0,16,0,16,0,16:12,41,11,10,0,2164,2175)
v1460s.assign    (var,data:0,8,0,8:6,26,8,9,0,1460,1470)
v1471s.assign    (var,data:0,16,0,16:8,26,10,9,0,1471,1481)
v1482s.assigncons(var,data:0,8,0,8:6,26,8,9,0,1482,1492)
v1493s.assigncons(var,data:0,16,0,16:6,26,8,9,0,1493,1503)
v3355s.clock     (name,freq:16,16:0,0,0,0,4,3355,3364)
v1225s.cons      (nam,val:0,8:0,0,0,0,0,1225,1236)
v1237s.cons      (nam,val:0,16:0,0,0,0,0,1237,1248)
v1828s.div       (rslt,arg1,arg2:0,8,0,8,0,8:11,136,11,11,0,1828,1840)
v1841s.div       (rslt,arg1,arg2:0,16,0,16,0,16:13,208,13,11,0,1841,1861)
v1569s.end       (::0,0,0,0,7,1569,1585)
v2248s.eq        (rslt,arg1,arg2:0,8,0,8,0,8:15,79,21,13,0,2248,2262)
v2263s.eq        (rslt,arg1,arg2:0,16,0,16,0,16:18,79,21,13,0,2263,2277)
v2446s.eventmark (signam,evaddr ::3,15,3,8,0,2446,2455)
v1366s.every     (nam::6,43,12,9,0,1366,1376)
v1184s.exitproc  (nam,cont::4,26,5,9,0,1184,1194)
v2012s.fadd      (rslt,arg1,arg2:0,24,0,24,0,24:0,0,0,14,6,2012,2027)
v1504s.fassign   (obj,src::0,0,0,0,0,1504,1510)
v2106s.fcons     (name,lmant,mmant,hmant,exp::0,0,0,10,0,2106,2117)
v2046s.fdiv      (rslt,arg1,arg2:0,32,0,32,0,32:0,0,0,14,6,2046,2061)
v2094s.fix       (rslt,arg:0,16,0,16:8,158,9,10,0,2094,2105)
v1425s.fixedwait (time:0,1000,0,10:10,43,10,4,17,1425,1443)
v1444s.fixedwait5(time:0,1000,0,10:8,27,8,4,0,1444,1459)
v2080s.float     (rslt,arg:0,8,0,8:7,158,9,12,5,2080,2093)
v1945s.fmul      (rslt,arg1,arg2:0,24,0,24,0,24:2,2,2,15,6,1945,1961)
v1377s.forcons   (lwr,upr,slab,elab:0,16,0,16,0,65535:7,44,11,15,0,1377,1393)
v1394s.forend    (slab,elab::2,17,2,8,0,1394,1403)
v1922s.fpack     (rslt:0,16:32,101,44,21,0,1922,1944)
v1978s.fsub      (rslt,arg1,arg2:0,24,0,24,0,24:0,0,0,14,6,1978,1993)
v1875s.funpack   (arg1,arg2:0,24,0,24:44,136,59,45,0,1875,1921)
v1286s.fvarset   (::24,0,0,0,0,1286,1309)
v2368s.ge        (rslt,arg1,arg2:0,8,0,8,0,8:15,79,21,13,0,2368,2382)
v2383s.ge        (rslt,arg1,arg2:0,16,0,16,0,16:18,79,21,13,0,2383,2397)
v2338s.gt        (rslt,arg1,arg2:0,8,0,8,0,8:15,79,21,13,0,2338,2352)
v2353s.gt        (rslt,arg1,arg2:0,16,0,16,0,16:18,79,21,13,0,2353,2367)
v1115s.heading   (::6,15,6,44,0,1115,1170)
v1777s.imul16    (rsh,rsl,arg1,arg2:0,16,0,16,0,16:14,156,14,11,0,1777,1789)
v1705s.imul8     (rslt,arg1,arg2:0,8,0,8,0,8:11,140,14,11,0,1705,1717)
v1756s.imulex8   (rslt,arg1,arg2:0,16,0,8,0,8:24,183,29,19,0,1756,1776)
v1790s.imulom16  (rslt,arg1,arg2:0,16,0,16,0,16:54,290,54,23,0,1790,1814)
v1718s.imulom8   (rslt,arg1,arg2:0,8,0,8,0,8:29,207,35,23,0,1718,1742)
v1815s.imulu16   (rslt,arg1,arg2:0,16,0,16,0,16:11,141,11,11,0,1815,1827)
v1743s.imulu8    (rslt,arg1,arg2:0,8,0,8,0,8:11,140,14,11,0,1743,1755)
v1310s.in        (::0,0,0,3,0,1310,1314)
```

```
v1543s.issuecond (signam:0,8,0,128:2,10,2,4,11,1543,1555)
v1556s.issuecond (signam:0,16,0,128:2,10,2,4,11,1556,1568)
v2574s.issueopcnd(signam:0,8,0,128:0,0,0,0,4,2574,2596)
v2651s.issuesepcd(s,v:0,8,0,128:4,23,5,5,7,2651,2667)
v2480s.issuevent (signam:0,8,0,128: 5,23,4,0,4,2480,2485)
v1347s.jmpf      (val,loc:0,8:8,36,10,9,0,1347,1357)
v1336s.jmpt      (val,loc:0,8:8,36,10,9,0,1336,1346)
v2308s.le        (rslt,arg1,arg2:0,8,0,8,0,8:15,79,21,13,0,2308,2322)
v2323s.le        (rslt,arg1,arg2:0,16,0,16,0,16:18,79,21,13,0,2323,2337)
v1358s.loc       (loc::1,3,1,6,0,1358,1365)
v2278s.lt        (rslt,arg1,arg2:0,8,0,8,0,8:15,79,21,13,0,2278,2292)
v2293s.lt        (rslt,arg1,arg2:0,16,0,16,0,16:18,79,21,13,0,2293,2307)
v1077s.main      (::0,0,0,2,4,1077,1087)
v1088s.monitor   (::24,43,28,3,0,1088,1114)
v1676s.mul       (rslt,arg1,arg2:0,8,0,8,0,8:17,114,3,11,0,1676,1688)
v1689s.mul       (rsh,rsl,arg1,arg2:0,16,0,16,0,16,0,16:18,191,4,14,0,1689,1704)
v2398s.ne        (rslt,arg1,arg2:0,8,0,8,0,8:15,79,21,13,0,2398,2412)
v2413s.ne        (rslt,arg1,arg2:0,16,0,16,0,16:18,79,21,13,0,2413,2435)
v2028s.nfadd      (rslt,arg1,arg2:0,24,0,24,0,24:11,286,17,16,6,2028,2045)
v2062s.nfdiv     (rslt,arg1,arg2:0,32,0,32,0,32:11,391,17,16,6,2062,2079)
v1962s.nfmul     (rslt,arg1,arg2:0,24,0,24,0,24:11,291,17,14,5,1962,1977)
v1994s.nfsub     (rslt,arg1,arg2:0,24,0,24,0,24:11,286,17,16,6,1994,2011)
v1315s.ni        (::0,0,0,3,0,1315,1327)
v2118s.nopck     (rslt:0,32:34,130,41,24,0,2118,2151)
v2200s.not       (rslt,arg:0,8,0,8:8,29,10,10,0,2200,2211)
v2212s.not       (rslt,arg:0,16,0,16,0,16:10,29,10,10,0,2212,2223)
v2176s.or        (rslt,arg1,arg2:0,8,0,8,0,8:10,41,12,10,0,2176,2187)
v2188s.or        (rslt,arg1,arg2:0,16,0,16,0,16:12,41,12,10,0,2188,2199)
v1328s.perform   (name::3,28,9,6,0,1328,1335)
v1174s.proc      (name::1,3,1,8,0,1174,1183)
v2643s.relayout  (s,v:0,2:0,0,0,0,4,2643,2650)
v3336s.restrans  (name,res:1000,10000:0,0,0,0,6,3336,3354)
v2798s.senscontac(s,b::0,0,0,0,6,2798,2810)
v1511s.sensecond (signam:0,8,0,128:2,10,2,4,11,1511,1526)
v1527s.sensecond (signam:0,16,0,128:2,10,2,4,11,1527,1542)
v2828s.sensepcond(s,b:0,8,0,128:5,25,4,5,11,2828,2842)
v2436s.sensevent (signam :1,1,0,8: 0,40,11,3,8,2436,2445)
v2811s.senshotct (s1,s2,v,m,b:10,60:0,0,0,9,8,2811,2827)
v2551s.sensopcond(signam:0,8,0,128:0,0,0,0,4,2551,2573)
v2542s.sensopevt (signam,sigret:1,1,0,8:0,0,0,0,5,2542,2550)
v1171s.start     (::0,0,0,0,0,1171,1173)
v1652s.sub       (rslt,arg1,arg2:0,8,0,8,0,8:9,38,3,10,0,1652,1663)
v1664s.sub       (rslt,arg1,arg2:0,16,0,16,0,16:10,38,3,10,0,1664,1675)
v1211s.tabaccp2  (fnc,task::16,85,23,12,0,1211,1224)
v1203s.tabend    (::3,15,5,6,0,1203,1210)
v1195s.tabent    (fnc,task::3,15,5,6,0,1195,1202)
v3314s.temp      (signam,hit,lot:-55,85,-55,85:0,0,0,0,12,3314,3335)
v1249s.var       (name,:0,8:1,0,0,3,0,1249,1260)
v1261s.var       (name,:0,16:2,0,0,3,0,1261,1272)
v1273s.var       (name,:0,24:4,0,0,3,0,1273,1285)
v1404s.whend     (whend,whtop::3,10,3,7,0,1404,1412)
v1413s.whilecon  (arg1,whend,whtop:0,8:11,42,12,10,0,1413,1424)
```

```
v2224s.xor       (rslt,arg1,arg2:0,8,0,8:10,41,12,10,0,2224,2235)
v2236s.xor       (rslt,arg1,arg2:0,16,0,16,0,16:10,41,12,10,0,2236,2247)
v0159 .end index
v0160com
v0161com*******************************************************************
v0162com
v0163com                Realization Library
v0164com
v0165com         for the Intel 8086 microprocessor
v0166com
v0167com  Author: Alan J. Cetel, LCDR, USN
v0168com          Naval Postgraduate School
v0169com          Monterey, CA
v0170com    Date: Fall 1983-Spring 1984
v0171com
v0172com*******************************************************************
v0173com
v0174com
v0175com*******************************************************************
v0176com
v0177com  hardware primitives
v0178com
v0179com*******************************************************************
v0180com
v0181com*******************************************************************
v0182com
v0183com  processor and support chips
v0184com
v0185com*******************************************************************
v0186com
v0187com*******************************************************************
v0188h.processor (::8,2500,1,32,33,188,222)
v0189com primitive to define central processing unit hardware w/o ndp
v0190com list = empty:empty:latency(clock in MHz),power(mW),number of chips
v0191com        calc,incl,addr
v0192com n.c. = no connection
v0193com ndp = numeric data processor
v0194begin htext
v0195 central processing unit
v0196  device:intel 8086 microprocessor(max-mode,no ndp),ic<icn>
v0197   connections:
v0198    pins 16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,39,38,37,36,35 = a(0:19)
v0199    pins 16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,39 = d(0:15)
v0200    pin 17 (nmi) = gnd
v0201    pin 18 (intr) = gnd
v0202    pin 19 = clk
v0203    pin 34 = bhe-bar
v0204    pin 33 (mn/max-bar) = gnd
v0205    pin 32 (rd-bar) = n.c.
v0206    pin 31 (rq-bar/gt0-bar) = n.c.
v0207    pin 30 (rq-bar/gt1-bar) = n.c.
v0208    pin 29 (lock-bar) = n.c.
```

```
v0209    pin 28 = s2-bar
v0210    pin 27 = s1-bar
v0211    pin 26 = s0-bar
v0212    pin 25 (qs0) = n.c.
v0213    pin 24 (qs1) = n.c.
v0214    pin 23 (test-bar) = gnd
v0215    pin 22 = ready
v0216    pin 21 = reset
v0217    pins 1,20 = gnd
v0218    pin 40 = +5v
v0219endtext
v0220calc icn = icn + 1
v0221call h.support   (::)
v0222com********************************************************************
v0223h.nprocessor(::5,2500,1,32,33,223,258)
v0224com primitive to define central processing unit hardware with ndp
v0225com list = empty:empty:latency(clock in MHz),power(mW),number of chips
v0226com        calc,incl,addr
v0227com n.c. = no connection
v0228attr clkper = 0.2
v0229begin htext
v0230 central processing unit
v0231  device:intel 8086 microprocessor(max-mode,with ndp),ic<icn>
v0232   connections:
v0233    pins 16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,39,38,37,36,35 = a(0:19)
v0234    pins 16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,39 = d(0:15)
v0235    pin 17 (nmi) = gnd
v0236    pin 18 (intr) = gnd
v0237    pin 19 = clk
v0238    pin 34 = bhe-bar
v0239    pin 33 (mn/max-bar) = gnd
v0240    pin 32 (rd-bar) = n.c.
v0241    pin 31 (rq-bar/gt0-bar) = n.c.
v0242    pin 30 (rq-bar/gt1-bar) = request/grant
v0243    pin 29 (lock-bar) = n.c.
v0244    pin 28 = s2-bar
v0245    pin 27 = s1-bar
v0246    pin 26 = s0-bar
v0247    pin 25  = qs0
v0248    pin 24  = qs1
v0249    pin 23 (test-bar) = gnd
v0250    pin 22 = ready
v0251    pin 21 = reset
v0252    pins 1,20 = gnd
v0253    pin 40 = +5v
v0254endtext
v0255calc icn = icn + 1
v0256call h.ndp       (::)
v0257call h.support   (::)
v0258com********************************************************************
v0259h.support   (::8,6610,15,33,0,259,569)
v0260com primitive to define support hardware for every realization
```

```
v0261com list = empty:empty:latency(clock in MHz),power(mW),number of chips
v0262com        calc,incl,addr
v0263begin htext
v0264 clock generator (0.125 us
v0265  device: intel 8284 clock gen and driver for 8086 cpu, ic<icn>
v0266  connections:
v0267   pin 1 (csync) = gnd
v0268   pin 3 (aen1-bar) = gnd
v0269   pin 4 (rdy1) = +5v
v0270   pin 5 (ready) = ready
v0271   pin 6 (rdy2) = gnd
v0272   pin 7 (aen2-bar) = gnd
v0273   pin 8 = clk
v0274   pin 10 = reset
v0275   pin 11 = res-bar
v0276   pin 13 (f/c-bar) = gnd
v0277   pin 14 (efi) = gnd
v0278   pin 15 (async-bar) = +5v
v0279endtext
v0280if flt .eg. 0 skip 4
v0281begin htext
v0282   pins 16,17 (xtal(1:2)) = device: 15 mhz crystal
v0283endtext
v0284skip 3
v0285begin htext
v0286   pins 16,17 (xtal(1:2)) = device: 24 mhz crystal
v0287endtext
v0288begin htext
v0289   pins 9 = gnd
v0290   pin 18 = +5v
v0291endtext
v0292calc icn = icn + 1
v0293begin htext
v0294 octal bus transceiver/data bits 0:7
v0295  device: intel 8286 octal bus transceiver, ic<icn>
v0296  connections:
v0297   pins 19,18,17,16,15,14,13,12 (db(0:7)) = db(0:7)
v0298   pin 1 (a0) = d(0)
v0299   pin 2 (a1) = d(1)
v0300   pin 3 (a2) = d(2)
v0301   pin 4 (a3) = d(3)
v0302   pin 5 (a4) = d(4)
v0303   pin 6 (a5) = d(5)
v0304   pin 7 (a6) = d(6)
v0305   pin 8 (a7) = d(7)
v0306   pin 9 (oe-bar) = .not. den
v0307   pin 11 (t) = dt/r-bar
v0308   pin 10 = gnd
v0309   pin 20 = +5v
v0310endtext
v0311calc icn = icn + 1
v0312begin htext
```

```
v0313 octal bus transceiver/data bits 8:15
v0314  device: intel 8286 octal bus transceiver, ic<icn>
v0315  connections:
v0316   pins 19,18,17,16,15,14,13,12 (db(0:7)) = db(8:15)
v0317   pin 1 (a0) = d(0)
v0318   pin 2 (a1) = d(1)
v0319   pin 3 (a2) = d(2)
v0320   pin 4 (a3) = d(3)
v0321   pin 5 (a4) = d(4)
v0322   pin 6 (a5) = d(5)
v0323   pin 7 (a6) = d(6)
v0324   pin 8 (a7) = d(7)
v0325   pin 9 (oe-bar) = .not. den
v0326   pin 11 (t) = dt/r-bar
v0327   pin 10 = gnd
v0328   pin 20 = +5v
v0329endtext
v0330calc icn = icn + 1
v0331begin htext
v0332 bus controller
v0333  device: intel 8288 bus controller for 8086 cpu, ic<icn>
v0334  connections:
v0335   pin 19 = s0-bar
v0336   pin 3 = s1-bar
v0337   pin 18 = s2-bar
v0338   pin 2 = clk
v0339   pin 5 = ale
v0340   pin 16 = den
v0341   pin 4 = dt/r-bar
v0342   pin 6 (ae-bar) = gnd
v0343   pin 1 (iob) = +5v
v0344   pin 7 = mrdc-bar
v0345   pin 9 = mwtc-bar
v0346   pin 11 = iowc-bar
v0347   pin 13 = iorc-bar
v0348   pin 14 = inta-bar
v0349   pin 15 (cen) = +5v
v0350   pin 10 = gnd
v0351   pin 20 = +5v
v0352endtext
v0353calc icn = icn + 1
v0354begin htext
v0355 octal latch/address bits 0:7
v0356  device: intel 8282 octal latch for 8086 cpu, ic<icn>
v0357  connections:
v0358   pins 1,2,3,4,5,6,7,8 (di(0:7)) = a(0:7)
v0359   pins 19,18,17,16,15,14,13,12 (do(0:7) = a(0:7)
v0360   pin 9 (oe-bar) = gnd
v0361   pin 11 (stb) = ale
v0362   pin 10 = gnd
v0363   pin 20 = +5v
v0364endtext
```

```
v0365calc icn = icn + 1
v0366begin htext
v0367 octal latch/address bits 8:15
v0368  device: intel 8282 octal latch for 8086 cpu, ic<icn>
v0369  connections:
v0370   pins 1,2,3,4,5,6,7,8 (di(0:7)) = a(8:15)
v0371   pins 19,18,17,16,15,14,13,12 (do(0:7)) = a(8:15)
v0372   pin 9 (oe-bar) = gnd
v0373   pin 11 (stb) = ale
v0374   pin 10 = gnd
v0375   pin 20 = +5v
v0376endtext
v0377calc icn = icn + 1
v0378begin htext
v0379 octal latch/address bits 16:19
v0380  device: intel 8282 octal latch for 8086 cpu, ic<icn>
v0381  connections:
v0382   pins 1,2,3,4 (di(0:3)) = a(16:19)
v0383   pins 19,18,17,16 (do(0:3) = a(16:19)
v0384   pins 5,6,7,8 (di(3:7)) = gnd
v0385   pin 9 (oe-bar) = gnd
v0386   pin 11 (stb) = ale
v0387   pin 10 = gnd
v0388   pin 20 = +5v
v0389endtext
v0390calc icn = icn + 1
v0391com primitive to define address decoder hardware, odd bytes
v0392begin htext
v0393 address decoder/address for memory select
v0394  device: intel 8205 1-of-8 binary decoder,ic<icn>
v0395  connections:
v0396   pin 15 (o(0)) = csu-bar(1)
v0397   pin 14 (o(1)) = csu-bar(2)
v0398   pin 13 (o(2)) = csu-bar(3)
v0399   pin 12 (o(3)) = csu-bar(4)
v0400   pin 11 (o(4)) = csu-bar(5)
v0401   pin 10 (o(5)) = csu-bar(6)
v0402   pin 9 (o(6)) = csu-bar(7)
v0403   pin 7 (o(7)) = csu-bar(8)
v0404   pin 1 = a(15)
v0405   pin 2 = a(16)
v0406   pin 3 = a(17)
v0407   pin 4 (e1-bar) = a(0)
v0408   pin 5 (e2-bar) = a(18)
v0409   pin 6 (e3) = .not. a(19)
v0410   pin 8 = gnd
v0411   pin 16 = +5v
v0412endtext
v0413calc icn = icn + 1
v0414begin htext
v0415 address decoder/address for memory select
v0416  device: intel 8205 1-of-8 binary decoder,ic<icn>
```

```
v0417  connections:
v0418   pin 15 (o(0)) = csu-bar(9)
v0419   pin 14 (o(1)) = csu-bar(10)
v0420   pin 13 (o(2)) = csu-bar(11)
v0421   pin 12 (o(3)) = csu-bar(12)
v0422   pin 11 (o(4)) = csu-bar(13)
v0423   pin 10 (o(5)) = csu-bar(14)
v0424   pin 9 (o(6)) = csu-bar(15)
v0425   pin 7 (o(7)) = csu-bar(16)
v0426   pin 1 = a(15)
v0427   pin 2 = a(16)
v0428   pin 3 = a(17)
v0429   pin 4 (e1-bar) = a(0)
v0430   pin 5 (e2-bar) = a(19)
v0431   pin 6 (e3) = a(18)
v0432   pin 8 = gnd
v0433   pin 16 = +5v
v0434endtext
v0435calc icn = icn + 1
v0436begin htext
v0437 address decoder/address for memory select
v0438  device: intel 8205 1-of-8 binary decoder,ic<icn>
v0439  connections:
v0440   pin 15 (o(0)) = csu-bar(17)
v0441   pin 14 (o(1)) = csu-bar(18)
v0442   pin 13 (o(2)) = csu-bar(19)
v0443   pin 12 (o(3)) = csu-bar(20)
v0444   pin 11 (o(4)) = csu-bar(21)
v0445   pin 10 (o(5)) = csu-bar(22)
v0446   pin 9 (o(6)) = csu-bar(23)
v0447   pin 7 (o(7)) = csu-bar(24)
v0448   pin 1 = a(15)
v0449   pin 2 = a(16)
v0450   pin 3 = a(17)
v0451   pin 4 (e1-bar) = a(0)
v0452   pin 5 (e2-bar) = a(18)
v0453   pin 6 (e3) = a(19)
v0454   pin 8 = gnd
v0455   pin 16 = +5v
v0456endtext
v0457calc icn = icn + 1
v0458begin htext
v0459 address decoder/address for memory select
v0460  device: intel 8205 1-of-8 binary decoder,ic<icn>
v0461  connections:
v0462   pin 15 (o(0)) = csu-bar(25)
v0463   pin 14 (o(1)) = csu-bar(26)
v0464   pin 13 (o(2)) = csu-bar(27)
v0465   pin 12 (o(3)) = csu-bar(28)
v0466   pin 11 (o(4)) = csu-bar(29)
v0467   pin 10 (o(5)) = csu-bar(30)
v0468   pin 9 (o(6)) = csu-bar(31)
```

```
v0469   pin 7 (o(7)) = csu-bar(32)
v0470   pin 1 = a(15)
v0471   pin 2 = a(16)
v0472   pin 3 = a(17)
v0473   pin 4 (e1-bar) = a(0)
v0474   pin 5 (e2-bar) = a(18)
v0475   pin 6 (e3) = .not. a(19)
v0476   pin 8 = gnd
v0477   pin 16 = +5v
v0478endtext
v0479calc icn = icn + 1
v0480com primitive to define address decoder hardware, even bytes
v0481begin htext
v0482 address decoder/address for memory select
v0483  device: intel 8205 1-of-8 binary decoder,ic<icn>
v0484  connections:
v0485   pin 15 (o(0)) = csl-bar(1)
v0486   pin 14 (o(1)) = csl-bar(2)
v0487   pin 13 (o(2)) = csl-bar(3)
v0488   pin 12 (o(3)) = csl-bar(4)
v0489   pin 11 (o(4)) = csl-bar(5)
v0490   pin 10 (o(5)) = csl-bar(6)
v0491   pin 9 (o(6)) = csl-bar(7)
v0492   pin 7 (o(7)) = csl-bar(8)
v0493   pin 1 = a(15)
v0494   pin 2 = a(16)
v0495   pin 3 = a(17)
v0496   pin 4 (e1-bar) = bhe-bar
v0497   pin 5 (e2-bar) = a(18)
v0498   pin 6 (e3) = .not. a(19)
v0499   pin 8 = gnd
v0500   pin 16 = +5v
v0501endtext
v0502calc icn = icn + 1
v0503begin htext
v0504 address decoder/address for memory select
v0505  device: intel 8205 1-of-8 binary decoder,ic<icn>
v0506  connections:
v0507   pin 15 (o(0)) = csl-bar(9)
v0508   pin 14 (o(1)) = csl-bar(10)
v0509   pin 13 (o(2)) = csl-bar(11)
v0510   pin 12 (o(3)) = csl-bar(12)
v0511   pin 11 (o(4)) = csl-bar(13)
v0512   pin 10 (o(5)) = csl-bar(14)
v0513   pin 9 (o(6)) = csl-bar(15)
v0514   pin 7 (o(7)) = csl-bar(16)
v0515   pin 1 = a(15)
v0516   pin 2 = a(16)
v0517   pin 3 = a(17)
v0518   pin 4 (e1-bar) = bhe-bar
v0519   pin 5 (e2-bar) = a(19)
v0520   pin 6 (e3) = a(18)
```

```
v0521   pin 8 = gnd
v0522   pin 16 = +5v
v0523endtext
v0524calc icn = icn + 1
v0525begin htext
v0526 address decoder/address for memory select
v0527  device: intel 8205 1-of-8 binary decoder,ic<icn>
v0528  connections:
v0529   pin 15 (o(0)) = csl-bar(17)
v0530   pin 14 (o(1)) = csl-bar(18)
v0531   pin 13 (o(2)) = csl-bar(19)
v0532   pin 12 (o(3)) = csl-bar(20)
v0533   pin 11 (o(4)) = csl-bar(21)
v0534   pin 10 (o(5)) = csl-bar(22)
v0535   pin 9 (o(6)) = csl-bar(23)
v0536   pin 7 (o(7)) = csl-bar(24)
v0537   pin 1 = a(15)
v0538   pin 2 = a(16)
v0539   pin 3 = a(17)
v0540   pin 4 (el-bar) = bhe-bar
v0541   pin 5 (e2-bar) = a(18)
v0542   pin 6 (e3) = a(19)
v0543   pin 8 = gnd
v0544   pin 16 = +5v
v0545endtext
v0546calc icn = icn + 1
v0547begin htext
v0548 address decoder/address for memory select
v0549  device: intel 8205 1-of-8 binary decoder,ic<icn>
v0550  connections:
v0551   pin 15 (o(0)) = csl-bar(25)
v0552   pin 14 (o(1)) = csl-bar(26)
v0553   pin 13 (o(2)) = csl-bar(27)
v0554   pin 12 (o(3)) = csl-bar(28)
v0555   pin 11 (o(4)) = csl-bar(29)
v0556   pin 10 (o(5)) = csl-bar(30)
v0557   pin 9 (o(6)) = csl-bar(31)
v0558   pin 7 (o(7)) = csl-bar(32)
v0559   pin 1 = a(15)
v0560   pin 2 = a(16)
v0561   pin 3 = a(17)
v0562   pin 4 (el-bar) = bhe-bar
v0563   pin 5 (e2-bar) = a(18)
v0564   pin 6 (e3) = .not. a(19)
v0565   pin 8 = gnd
v0566   pin 16 = +5v
v0567endtext
v0568calc icn = icn + 1
v0569com*****************************************************************
v0570h.ndp       (::8,3000,1,31,0,570,602)
v0571com primitive to define numeric data processing hardware
v0572com list = empty:empty:latency(clock in MHz),power(mW),number of chips
```

```
v0573com        calc,incl,addr
v0574com n.c. = no connection
v0575begin htext
v0576 numeric data processor
v0577  device:intel 8087 numeric data processor,ic<icn>
v0578   connections:
v0579    pins 16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,39,38,37,36,35 = a(0:19)
v0580    pins 16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,39 = d(0:15)
v0581    pin 17 = n.c.
v0582    pin 18 = n.c.
v0583    pin 19 = clk
v0584    pin 34 = bhe-bar
v0585    pin 33 (rq/gt1-bar) = n.c.
v0586    pin 32 (int) = gnd
v0587    pin 31 (rq-bar/gt0-bar) = request/grant
v0588    pin 30 = n.c.
v0589    pin 29 = n.c.
v0590    pin 28 = s2-bar
v0591    pin 27 = s1-bar
v0592    pin 26 = s0-bar
v0593    pin 25 (qs0) = n.c.
v0594    pin 24 (qs1) = n.c.
v0595    pin 23 (busy) = test
v0596    pin 22 = ready
v0597    pin 21 = reset
v0598    pins 1,20 = gnd
v0599    pin 40 = +5v
v0600endtext
v0601calc icn = icn + 1
v0602com****************************************************************
v0603h.eprom     (cnt::0,150,2,15,0,603,632)
v0604com 128k eprom
v0605com list=select:empty:latency,power,no. of chips,calc,incl,addr
v0606begin htext
v0607 128k eprom lower half of page, <cnt>
v0608  device: intel 27128 128k (16k*8) uv eprom, ic<icn>
v0609  connections:
v0610   pins 10,9,8,7,6,5,4,3,25,24,21,23,2,26 (a(0:13)) = a(1:14)
v0611   pins 11,12,13,15,16,17,18,19 (io(1:8)) = db(0:7)
v0612   pin 20 (ce-bar) = csl-bar<cnt>
v0613   pin 22 (oe-bar) = gnd
v0614   pin 27 (pgm-bar) = gnd
v0615   pins 1,28 = +5v
v0616   pin 14 = gnd
v0617endtext
v0618calc icn = icn + 1
v0619begin htext
v0620 128k eprom upper half of page, <cnt>
v0621  device: intel 27128 128k (16k*8) uv eprom, ic<icn>
v0622  connections:
v0623   pins 10,9,8,7,6,5,4,3,25,24,21,23,2,26 (a(0:13)) = a(1:14)
v0624   pins 11,12,13,15,16,17,18,19 (io(1:8)) = db(8:15)
```

```
v0625   pin 20 (ce-bar) = csu-bar<cnt>
v0626   pin 22 (oe-bar) = gnd
v0627   pin 27 (pgm-bar) = gnd
v0628   pins 1,28 = +5v
v0629   pin 14 = gnd
v0630endtext
v0631calc icn = icn + 1
v0632com*****************************************************************
v0633h.ioram     (::0,200,4,16,0,633,692)
v0634com 4k input/output static ram
v0635com list=select:empty:latency,power,no. of chips,calc,incl,addr
v0636begin htext
v0637 4k i/o ram
v0638  device: intel 2142-2 4k (1k*4) static ram,ic<icn>
v0639  connections:
v0640   pins 6,7,8,4,3,2,1,19,18,17 (a(0:9)) = a(1:10)
v0641   pins 15,14,13,12 (dio(1:4)) = d(1:4)
v0642   pin 11 (we-bar) = iowc-bar
v0643   pin 9 (cs1-bar) = csl1
v0644   pin 16 (od) = .not. iorc-bar
v0645   pin 5 (cs2-bar) = gnd
v0646   pin 20 = +5v
v0647   pin 10 = gnd
v0648endtext
v0649calc icn = icn + 1
v0650begin htext
v0651 4k i/o ram
v0652  device: intel 2142-2 4k (1k*4) static ram,ic<icn>
v0653  connections:
v0654   pins 6,7,8,4,3,2,1,19,18,17 (a(0:9)) = a(1:10)
v0655   pins 15,14,13,12 (dio(1:4)) = d(5:8)
v0656   pin 11 (we-bar) = iowc-bar
v0657   pin 9 (cs1-bar) = csl1
v0658   pin 16 (od) = .not. iorc-bar
v0659   pin 5 (cs2-bar) = gnd
v0660   pin 20 = +5v
v0661   pin 10 = gnd
v0662endtext
v0663calc icn = icn + 1
v0664begin htext
v0665 4k i/o ram
v0666  device: intel 2142-2 4k (1k*4) static ram,ic<icn>
v0667  connections:
v0668   pins 6,7,8,4,3,2,1,19,18,17 (a(0:9)) = a(1:10)
v0669   pins 15,14,13,12 (dio(1:4)) = d(9:12)
v0670   pin 11 (we-bar) = iowc-bar
v0671   pin 9 (cs1-bar) = csul
v0672   pin 16 (od) = .not. iorc-bar
v0673   pin 5 (cs2-bar) = gnd
v0674   pin 20 = +5v
v0675   pin 10 = gnd
v0676endtext
```

```
v0677calc icn = icn + 1
v0678begin htext
v0679 4k i/o ram
v0680  device: intel 2142-2 4k (1k*4) static ram,ic<icn>
v0681  connections:
v0682   pins 6,7,8,4,3,2,1,19,18,17 (a(0:9)) = a(1:10)
v0683   pins 15,14,13,12 (dio(1:4)) = d(13:16)
v0684   pin 11 (we-bar) = iowc-bar
v0685   pin 9 (cs1-bar) = csu1
v0686   pin 16 (od) = .not. iorc-bar
v0687   pin 5 (cs2-bar) = gnd
v0688   pin 20 = +5v
v0689   pin 10 = gnd
v0690endtext
v0691calc icn = icn + 1
v0692com****************************************************************
v0693h.ram       (cnt::0,720,18,15,0,693,926)
v0694com 16k static ram
v0695com list=select:empty:latency,power,no. of chips,calc,incl,addr
v0696begin htext
v0697 16k ram all bits plus drivers, <cnt>
v0698  device:intel 2167-10 16k (16k*1) static ram,ic<icn>
v0699  connections:
v0700   pins 1,2,3,4,5,6,7,13,14,15,16,17,18,19 (a(0:13)) = a(1:14)
v0701   pin 12 (din) = d(0)
v0702   pin 9 (we-bar) = mwtc-bar
v0703   pin 8 (do) = d(0)
v0704   pin 11 (cs-bar) = csl<cnt>
v0705   pin 20 = +5v
v0706   pin 10 = gnd
v0707endtext
v0708calc icn = icn + 1
v0709begin htext
v0710 16k ram all bits plus drivers, <cnt>
v0711  device:intel 2167-10 16k (16k*1) static ram,ic<icn>
v0712  connections:
v0713   pins 1,2,3,4,5,6,7,13,14,15,16,17,18,19 (a(0:13)) = a(1:14)
v0714   pin 12 (din) = d(1)
v0715   pin 9 (we-bar) = mwtc-bar
v0716   pin 8 (do) = d(1)
v0717   pin 11 (cs-bar) = csl<cnt>
v0718   pin 20 = +5v
v0719   pin 10 = gnd
v0720endtext
v0721calc icn = icn + 1
v0722begin htext
v0723 16k ram all bits plus drivers, <cnt>
v0724  device:intel 2167-10 16k (16k*1) static ram,ic<icn>
v0725  connections:
v0726   pins 1,2,3,4,5,6,7,13,14,15,16,17,18,19 (a(0:13)) = a(1:14)
v0727   pin 12 (din) = d(2)
v0728   pin 9 (we-bar) = mwtc-bar
```

```
v0729   pin 8 (do) = d(2)
v0730   pin 11 (cs-bar) = csl<cnt>
v0731   pin 20 = +5v
v0732   pin 10 = gnd
v0733endtext
v0734calc icn = icn + 1
v0735begin htext
v0736 16k ram all bits plus drivers, <cnt>
v0737  device:intel 2167-10 16k (16k*1) static ram,ic<icn>
v0738  connections:
v0739   pins 1,2,3,4,5,6,7,13,14,15,16,17,18,19 (a(0:13)) = a(1:14)
v0740   pin 12 (din) = d(3)
v0741   pin 9 (we-bar) = mwtc-bar
v0742   pin 8 (do) = d(3)
v0743   pin 11 (cs-bar) = csl<cnt>
v0744   pin 20 = +5v
v0745   pin 10 = gnd
v0746endtext
v0747calc icn = icn + 1
v0748begin htext
v0749 16k ram all bits plus drivers, <cnt>
v0750  device:intel 2167-10 16k (16k*1) static ram,ic<icn>
v0751  connections:
v0752   pins 1,2,3,4,5,6,7,13,14,15,16,17,18,19 (a(0:13)) = a(1:14)
v0753   pin 12 (din) = d(4)
v0754   pin 9 (we-bar) = mwtc-bar
v0755   pin 8 (do) = d(4)
v0756   pin 11 (cs-bar) = csl<cnt>
v0757   pin 20 = +5v
v0758   pin 10 = gnd
v0759endtext
v0760calc icn = icn + 1
v0761begin htext
v0762 16k ram all bits plus drivers, <cnt>
v0763  device:intel 2167-10 16k (16k*1) static ram,ic<icn>
v0764  connections:
v0765   pins 1,2,3,4,5,6,7,13,14,15,16,17,18,19 (a(0:13)) = a(1:14)
v0766   pin 12 (din) = d(5)
v0767   pin 9 (we-bar) = mwtc-bar
v0768   pin 8 (do) = d(5)
v0769   pin 11 (cs-bar) = csl<cnt>
v0770   pin 20 = +5v
v0771   pin 10 = gnd
v0772endtext
v0773calc icn = icn + 1
v0774begin htext
v0775 16k ram all bits plus drivers, <cnt>
v0776  device:intel 2167-10 16k (16k*1) static ram,ic<icn>
v0777  connections:
v0778   pins 1,2,3,4,5,6,7,13,14,15,16,17,18,19 (a(0:13)) = a(1:14)
v0779   pin 12 (din) = d(6)
v0780   pin 9 (we-bar) = mwtc-bar
```

```
v0781   pin 8 (do) = d(6)
v0782   pin 11 (cs-bar) = csl<cnt>
v0783   pin 20 = +5v
v0784   pin 10 = gnd
v0785endtext
v0786calc icn = icn + 1
v0787begin htext
v0788 16k ram all bits plus drivers, <cnt>
v0789  device:intel 2167-10 16k (16k*1) static ram,ic<icn>
v0790  connections:
v0791   pins 1,2,3,4,5,6,7,13,14,15,16,17,18,19 (a(0:13)) = a(1:14)
v0792   pin 12 (din) = d(7)
v0793   pin 9 (we-bar) = mwtc-bar
v0794   pin 8 (do) = d(7)
v0795   pin 11 (cs-bar) = csl<cnt>
v0796   pin 20 = +5v
v0797   pin 10 = gnd
v0798endtext
v0799calc icn = icn + 1
v0800begin htext
v0801 16k ram all bits plus drivers, <cnt>
v0802  device:intel 2167-10 16k (16k*1) static ram,ic<icn>
v0803  connections:
v0804   pins 1,2,3,4,5,6,7,13,14,15,16,17,18,19 (a(0:13)) = a(1:14)
v0805   pin 12 (din) = d(8)
v0806   pin 9 (we-bar) = mwtc-bar
v0807   pin 8 (do) = d(8)
v0808   pin 11 (cs-bar) = csu<cnt>
v0809   pin 20 = +5v
v0810   pin 10 = gnd
v0811endtext
v0812calc icn = icn + 1
v0813begin htext
v0814 16k ram all bits plus drivers, <cnt>
v0815  device:intel 2167-10 16k (16k*1) static ram,ic<icn>
v0816  connections:
v0817   pins 1,2,3,4,5,6,7,13,14,15,16,17,18,19 (a(0:13)) = a(1:14)
v0818   pin 12 (din) = d(9)
v0819   pin 9 (we-bar) = mwtc-bar
v0820   pin 8 (do) = d(9)
v0821   pin 11 (cs-bar) = csu<cnt>
v0822   pin 20 = +5v
v0823   pin 10 = gnd
v0824endtext
v0825calc icn = icn + 1
v0826begin htext
v0827 16k ram all bits plus drivers, <cnt>
v0828  device:intel 2167-10 16k (16k*1) static ram,ic<icn>
v0829  connections:
v0830   pins 1,2,3,4,5,6,7,13,14,15,16,17,18,19 (a(0:13)) = a(1:14)
v0831   pin 12 (din) = d(10)
v0832   pin 9 (we-bar) = mwtc-bar
```

```
v0833   pin 8 (do) = d(10)
v0834   pin 11 (cs-bar) = csu<cnt>
v0835   pin 20 = +5v
v0836   pin 10 = gnd
v0837endtext
v0838calc icn = icn + 1
v0839begin htext
v0840 16k ram all bits plus drivers, <cnt>
v0841  device:intel 2167-10 16k (16k*1) static ram,ic<icn>
v0842  connections:
v0843   pins 1,2,3,4,5,6,7,13,14,15,16,17,18,19 (a(0:13)) = a(1:14)
v0844   pin 12 (din) = d(11)
v0845   pin 9 (we-bar) = mwtc-bar
v0846   pin 8 (do) = d(11)
v0847   pin 11 (cs-bar) = csu<cnt>
v0848   pin 20 = +5v
v0849   pin 10 = gnd
v0850endtext
v0851calc icn = icn + 1
v0852begin htext
v0853 16k ram all bits plus drivers, <cnt>
v0854  device:intel 2167-10 16k (16k*1) static ram,ic<icn>
v0855  connections:
v0856   pins 1,2,3,4,5,6,7,13,14,15,16,17,18,19 (a(0:13)) = a(1:14)
v0857   pin 12 (din) = d(12)
v0858   pin 9 (we-bar) = mwtc-bar
v0859   pin 8 (do) = d(12)
v0860   pin 11 (cs-bar) = csu<cnt>
v0861   pin 20 = +5v
v0862   pin 10 = gnd
v0863endtext
v0864calc icn = icn + 1
v0865begin htext
v0866 16k ram all bits plus drivers, <cnt>
v0867  device:intel 2167-10 16k (16k*1) static ram,ic<icn>
v0868  connections:
v0869   pins 1,2,3,4,5,6,7,13,14,15,16,17,18,19 (a(0:13)) = a(1:14)
v0870   pin 12 (din) = d(13)
v0871   pin 9 (we-bar) = mwtc-bar
v0872   pin 8 (do) = d(13)
v0873   pin 11 (cs-bar) = csu<cnt>
v0874   pin 20 = +5v
v0875   pin 10 = gnd
v0876endtext
v0877calc icn = icn + 1
v0878begin htext
v0879 16k ram all bits plus drivers, <cnt>
v0880  device:intel 2167-10 16k (16k*1) static ram,ic<icn>
v0881  connections:
v0882   pins 1,2,3,4,5,6,7,13,14,15,16,17,18,19 (a(0:13)) = a(1:14)
v0883   pin 12 (din) = d(14)
v0884   pin 9 (we-bar) = mwtc-bar
```

```
v0885   pin 8 (do) = d(14)
v0886   pin 11 (cs-bar) = csu<cnt>
v0887   pin 20 = +5v
v0888   pin 10 = gnd
v0889endtext
v0890calc icn = icn + 1
v0891begin htext
v0892 16k ram all bits plus drivers, <cnt>
v0893  device:intel 2167-10 16k (16k*1) static ram,ic<icn>
v0894  connections:
v0895   pins 1,2,3,4,5,6,7,13,14,15,16,17,18,19 (a(0:13)) = a(1:14)
v0896   pin 12 (din) = d(15)
v0897   pin 9 (we-bar) = mwtc-bar
v0898   pin 8 (do) = d(15)
v0899   pin 11 (cs-bar) = csu<cnt>
v0900   pin 20 = +5v
v0901   pin 10 = gnd
v0902endtext
v0903calc icn = icn + 1
v0904begin htext
v0905 address buffers (lower byte d(0:7))
v0906  device:amd 74ls244 octal three-state buffer,ic<icn>
v0907  connections:
v0908   pins 1,19 (enable-bar) = gnd
v0909   pins 2,4,6,8 (1a1:4) = a(0:3)
v0910   pins 18,16,14,12 (1y1:4) = a(0:3)
v0911   pins 11,13,15,17 (2a1:4) = a(4:7)
v0912   pins 9,7,5,3 (2y1:4) = a(4:7)
v0913endtext
v0914calc icn = icn +1
v0915begin htext
v0916 address buffers (upper byte d(8:13))
v0917  device:amd 74ls244 octal three-state buffer,ic<icn>
v0918  connections:
v0919   pins 1,19 (enable-bar) = gnd
v0920   pins 2,4,6,8 (1a1:4) = a(8:11)
v0921   pins 18,16,14,12 (1y1:4) = a(8:11)
v0922   pins 11,13,15,17 (2a1:4) = a(12:13)
v0923   pins 9,7,5,3 (2y1:4) = a(12:13)
v0924endtext
v0925calc icn = icn +1
v0926com****************************************************************
v0927h.drvr      (::0,135,2,15,0,927,954)
v0928com primitive to implement address line drivers as more banks of
v0929com memory are added
v0930com list = empty:empty:latency,power(mW),number of chips,calc,incl,
v0931com        addr
v0932begin htext
v0933 address buffers
v0934  device:amd 74ls244 octal three-state buffer,ic<icn>
v0935  connections:
v0936   pins 1,19 (enable-bar) = gnd
```

```
v0937   pins 2,4,6,8 (1a1:4) = a(0:3)
v0938   pins 18,16,14,12 (1y1:4) = a(0:3)
v0939   pins 11,13,15,17 (2a1:4) = a(4:7)
v0940   pins 9,7,5,3 (2y1:4) = a(4:7)
v0941endtext
v0942calc icn = icn +1
v0943begin htext
v0944 address buffers
v0945  device:amd 74ls244 octal three-state buffer,ic<icn>
v0946  connections:
v0947   pins 1,19 (enable-bar) = gnd
v0948   pins 2,4,6,8 (1a1:4) = a(8:11)
v0949   pins 18,16,14,12 (1y1:4) = a(8:11)
v0950   pins 11,13,15,17 (2a1:4) = a(12:13)
v0951   pins 9,7,5,3 (2y1:4) = a(12:13)
v0952endtext
v0953calc icn = icn +1
v0954com****************************************************************
v0955h.sensecond (signam,inport:0,8,0,1024:2,500,1,18,0,955,974)
v0956com primitive to define condition-type input hardware
v0957com list=source:min/max-input-lines,min/max-conditions:latency
v0958com list=power,number of chips,calc,incl,addr
v0959begin htext
v0960 condition mode input interface hardware to sense signal <signam>
v0961  device:intel 8212 8 bit i/o port,ic <icn>
v0962  connections:
v0963   pins 3,5,7,9,16,18,20,22(di(1:8)) = <signam>(1:8) remainder to
v0964                                                      ground
v0965   pins 4,6,8,10,15,17,19,21(do(1:8)) = db(1:8)
v0966   pin 2 (md) = gnd
v0967   pin 11 (stb) = gnd
v0968   pin 1 (ds1-bar) = .not. (decode a(0:7) value <inport>)
v0969   pin 13 (ds2) = inp .and. dbin
v0970   pin 24 = +5v
v0971   pin 12 = gnd
v0972endtext
v0973calc icn = icn + 1
v0974com****************************************************************
v0975h.sensecond (signam,inport:0,16,0,1024:2,1000,2,18,0,975,1009)
v0976com primitive to define condition-type input hardware
v0977com list=source:min/max-input-lines,min/max-conditions:latency
v0978com list=power,number of chips,calc,incl,addr
v0979begin htext
v0980 condition mode input interface hardware to sense signal <signam>
v0981  device:intel 8212 8 bit i/o port,ic <icn>
v0982  connections:
v0983   pins 3,5,7,9,16,18,20,22(di(1:8)) = <signam>(1:8) remainder to
v0984                                                      ground
v0985   pins 4,6,8,10,15,17,19,21(do(1:8)) = db(1:8)
v0986   pin 2 (md) = gnd
v0987   pin 11 (stb) = gnd
v0988   pin 1 (ds1-bar) = .not. (decode a(0:7) value <inport>)
```

```
v0989   pin 13 (ds2) = inp .and. dbin
v0990   pin 24 = +5v
v0991   pin 12 = gnd
v0992endtext
v0993calc icn = icn + 1
v0994begin htext
v0995 condition mode input interface hardware to sense signal <signam>
v0996  device:intel 8212 8 bit i/o port,ic <icn>
v0997  connections:
v0998   pins 3,5,7,9,16,18,20,22(di(9:16)) = <signam>(9:16) remainder to
v0999                                                        ground
v1000   pins 4,6,8,10,15,17,19,21(do(9:16)) = db(9:16)
v1001   pin 2 (md) = gnd
v1002   pin 11 (stb) = gnd
v1003   pin 1 (ds1-bar) = .not. (decode a(8:15) value <inport>)
v1004   pin 13 (ds2) = inp .and. dbin
v1005   pin 24 = +5v
v1006   pin 12 = gnd
v1007endtext
v1008calc icn = icn + 1
v1009com****************************************************************
v1010h.memory    (:0,1048576:0,0,0,9,7,1010,1045)
v1011com primitive to implement memory hardware
v1012com implement ram memory at bottom of address space
v1013com list = empty:max address space:latency,power,number of chips,
v1014com        calc, incl,addr
v1015com check to insure that rom and ram space do not overlap
v1016if ramptr .lt. romptr skip 2
v1017call h.herr      (::)
v1018skip 25
v1019calc mem = 0
v1020if mem .ge. ramptr skip 8
v1021incl h.ram       (<cnt>::)
v1022calc scrtch = scrtch + 1
v1023calc cnt = cnt + 1
v1024if scrtch .lt. 4 skip 2
v1025call h.drvr      (::)
v1026calc scrtch = 1
v1027calc mem = mem + 16384
v1028skip - 9
v1029com end loop
v1030com implement rom memory at top of address space
v1031calc scrtch = 1
v1032calc cnt = 32
v1033calc mem = 1048576
v1034if mem .lt. romptr skip 8
v1035calc mem = mem - 16384
v1036call h.eprom     (<cnt>:)
v1037calc cnt = cnt - 1
v1038calc scrtch = scrtch + 1
v1039if scrtch .lt. 5 skip 2
v1040call h.drvr      (::)
```

```
v1041calc scrtch = 1
v1042skip - 9
v1043call h.ioram     (::)
v1044com end loop
v1045com****************************************************************
v1046h.herr      (::0,0,0,0,0,1046,1076)
v1047com primitive to warn system designer fo a rom area/ram area
v1048com overlap.  this is an implementation in the library only
v1049begin htext
v1050*******************************************************************
v1051*******************************************************************
v1052**                                                               **
v1053**        w a r n i n g  - - -  m e m o r y    e r r o r         **
v1054**                                                               **
v1055**    this design has resulted in an overflow of rom area into  **
v1056**    ram area and will not be able to be implemented unless     **
v1057**    some action is taken to reduce rom or ram space            **
v1058**                                                               **
v1059*******************************************************************
v1060*******************************************************************
v1061endtext
v1062com****************************************************************
v1063com
v1064com****************************************************************
v1065com
v1066com  software primitives
v1067com
v1068com****************************************************************
v1069com
v1070com****************************************************************
v1071com
v1072com  monitor
v1073com
v1074com****************************************************************
v1075com
v1076com****************************************************************
v1077s.main      (::0,0,0,2,4,1077,1087)
v1078com primitive to define software initalization
v1079calc evpnt = -1
v1080if flt .eq. 1 skip 2
v1081incl h.processor (::)
v1082skip 1
v1083incl h.nprocessor(::)
v1084calc romptr = 1048560
v1085calc ramptr = 1024
v1086call s.heading   (::)
v1087com****************************************************************
v1088s.monitor   (::24,43,28,3,0,1088,1114)
v1089com primitive to define the p2 monitor as controller supervisor
v1090com list=empty:empty:storage,time,ext,calc,incl,addr
v1091calc ramptr = ramptr - 1
v1092com
```

```
v1093begin stext
v1094;
v1095;     -   monitor  section  -
v1096;
v1097@spvsr:      mov    AX,@table        ;initalize table pointer
v1098             mov    @pntr,AX         ; to beginning
v1099@mlop:       mov    BX,@pntr         ;monitor loop
v1100             inc    BX
v1101             inc    BX
v1102             inc    BX
v1103             mov    @pntr,BX
v1104             jmp    BX
v1105;
v1106;     -  data  section  -
v1107;
v1108             org    <ramptr>
v1109@pntr:       dw     0                ;table entry address pointer
v1110             org    <romptr>         ;rom address pointer
v1111@table:      dw     @pntr            ;table header (define top)
v1112endtext
v1113calc romptr = romptr + 2
v1114com******************************************************************
v1115s.heading   (::6,15,6,44,0,1115,1170)
v1116com subroutine to define software heading
v1117com list=empty:empty:storage,time,ext,calc,incl,addr
v1118begin stext
v1119;
v1120;=========================================
v1121;
v1122;     - intel 8086 realization -
v1123;
v1124;=========================================
v1125;
v1126;    idsec
v1127;    idsec
v1128;    idsec
v1129;
v1130sys14        equu   0D000H
v1131sys13        equ    0C000H
v1132sys12        equ    0B000H
v1133sys11        equ    0A000H
v1134sys10        equ    9000H
v1135sys9         equ    8000H
v1136sys8         equ    7000H
v1137sys7         equ    6000H
v1138sys6         equ    5000H
v1139sys5         equ    4000H
v1140sys4         equ    3000H
v1141sys3         equ    2000H
v1142sys2         equ    1000H
v1143;
v1144;
```

```
v1145;this routine allows for a 1K stack.  a stack that grows larger than
v1146;this will overflow into the data segment.  to allow a larger stack
v1147;the ramptr global primitive in s.main must be set to the required
v1148;value.  this method overlaps 64K of stack segment and 64K of data
v1149;segment.
v1150;
v1151            org    <romptr>        ;rom address pointer
v1152            mov    DX,03FFH        ;set data segment base address
v1153            mov    DS,DX           ;to 1024
v1154            mov    DX,0000H        ;set stack segment base address
v1155            mov    SS,DX           ;to 0000H
v1156            jmp    0F000H          ;jump to low address of highest
v1157                                   ;64K block
v1158endtext
v1159calc romptr = romptr - 13
v1160begin stext
v1161            org    <romptr>        ;rom address pointer
v1162            jmp    0E000H          ;to bottom of 2nd high 64K
v1163                                   ;block
v1164endtext
v1165calc romptr = 983040
v1166begin stext
v1167            org    <ramptr>
v1168endtext
v1169calc romptr = romptr + 6
v1170com****************************************************************
v1171s.start     (::0,0,0,0,0,1171,1173)
v1172com dummy start primitive
v1173com****************************************************************
v1174s.proc      (name::1,3,1,8,0,1174,1183)
v1175com routine to define procedure entry point
v1176com list=procname:empty:storage,time,ext,calc,incl,addr
v1177begin stext
v1178;
v1179;procedure <name>
v1180@<name>:    nop                     ;entry point for <name>
v1181endtext
v1182calc romptr = romptr + 1
v1183com****************************************************************
v1184s.exitproc  (nam,cont::4,26,5,9,0,1184,1194)
v1185com routine to close procedure and reset associated contingency
v1186com list=procname,contname:empty:storage,time,ext,calc,incl,addr
v1187if <cont> .eq. 0 skip 6
v1188begin stext
v1189;exit procedure
v1190            mov    <cont>,0         ;set <cont> to zero
v1191            ret                     ;return to monitor, exit <nam>
v1192endtext
v1193calc romptr = romptr + 4
v1194com****************************************************************
v1195s.tabent    (fnc,task::3,15,5,6,0,1195,1202)
v1196com primitive to add one entry to monitor table
```

```
v1197com list=fncname,taskname:empty:storage,time,ext,calc,incl,addr
v1198begin stext
v1199            jmp    @t<fnc>          ;test for contingency <fnc>
v1200endtext
v1201calc romptr = romptr + 3
v1202com****************************************************************
v1203s.tabend    (::3,15,5,6,0,1203,1210)
v1204com routine to define end of monitor table
v1205com list=empty:empty:storage,time,ext,calc,incl,addr
v1206begin stext
v1207            jmp    @spvsr           ;go to start of table
v1208endtext
v1209calc romptr = romptr + 3
v1210com****************************************************************
v1211s.tabaccp2  (fnc,task::16,85,23,12,0,1211,1224)
v1212com subroutine to add routine access for contingency/task pair
v1213com list=empty:empty:storage,time,ext,calc,incl,addr
v1214begin stext
v1215;
v1216@t<fnc>:    call   @<fnc>           ;execute contingency code <fnc>
v1217            cmp    <fnc>,1          ;compare contingency result to
v1218                                    ;true flag (1)
v1219            jnz    $ + 5            ;if false do not execute <task>
v1220            call   @<task>          ;execute task <task> if true
v1221            jmp    @mlop            ;return to monitor
v1222endtext
v1223calc romptr = romptr + 16
v1224com****************************************************************
v1225s.cons      (nam,val:0,8:0,0,0,0,0,1225,1236)
v1226com primitive to define data constants
v1227com must be declared before value is used in program
v1228com for for binding at assembly time
v1229com list = dataname,value:precision:storage,time,ext,calc,incl,
v1230com        addr
v1231begin stext
v1232;define 8-bit data constant
v1233<nam>      equu    <val>
v1234endtext
v1235com no rom or ram used
v1236com****************************************************************
v1237s.cons      (nam,val:0,16:0,0,0,0,0,1237,1248)
v1238com primitive to define data constants
v1239com must be declared before value is used in program
v1240com for for binding at assembly time
v1241com list = dataname,value:precision:storage,time,ext,calc,incl,
v1242com        addr
v1243begin stext
v1244;define 16-bit data constant
v1245<nam>      equ     <val>
v1246endtext
v1247com no rom or ram used
v1248com****************************************************************
```

```
v1249s.var       (name,:0,8:1,0,0,3,0,1249,1260)
v1250com routine to define storage for 8 bit integer variable
v1251com list=name:precision:storage,time,ext,calc,incl,addr
v1252calc ramptr = ramptr - 1
v1253begin stext
v1254;define 8-bit storage
v1255            org    <ramptr>         ;8 bit variable <name> in ram
v1256<name>:     db     0
v1257            org    <romptr>        ;rom address pointer
v1258endtext
v1259calc ramptr = ramptr + 1
v1260com****************************************************************
v1261s.var       (name,:0,16:2,0,0,3,0,1261,1272)
v1262com routine to define storage for 16 bit integer variable
v1263com list=name:precision:storage,time,ext,calc,incl,addr
v1264calc ramptr = ramptr - 1
v1265begin stext
v1266;define 16-bit storage
v1267            org    <ramptr>         ;16 bit variable <name> in ram
v1268<name>:     dw     0
v1269            org    <romptr>        ;rom address pointer
v1270endtext
v1271calc ramptr = ramptr + 2
v1272com****************************************************************
v1273s.var       (name,:0,24:4,0,0,3,0,1273,1285)
v1274com routine to define storage for 24 bit integer variable
v1275com list=name:precision:storage,time,ext,calc,incl,addr
v1276calc ramptr = ramptr - 1
v1277begin stext
v1278;define 24-bit storage
v1279            org    <ramptr>         ;24 bit variable <name> in ram
v1280<name>:     dw     0
v1281            dw     0
v1282            org    <romptr>        ;rom address pointer
v1283endtext
v1284calc ramptr = ramptr + 4
v1285com****************************************************************
v1286s.fvarset   (::24,0,0,0,0,1286,1309)
v1287com routine to establish ram storage for floating point operands
v1288com and result
v1289com list=empty:empty:storage,time,ext,calc,incl,addr
v1290begin stext
v1291;establish ram storage for floating point
v1292            org    <ramptr>
v1293lman1:      ds     1
v1294uman1:      ds     1
v1295exp1:      ds     1
v1296sign1:      ds     1
v1297;
v1298lman2:      ds     1
v1299uman2:      ds     1
v1300exp2:      ds     1
```

```
v1301sign2:      ds     1
v1302;
v1303lman3:      ds     1
v1304uman3:      ds     1
v1305exp3:       ds     1
v1306sign3:      ds     1
v1307            org    <romptr>        ;rom address pointer
v1308endtext
v1309com****************************************************************
v1310s.in        (::0,0,0,3,0,1310,1314)
v1311com routine to set the timed block flag
v1312com list=empty:empty:storage,time,ext,calc,incl,addr
v1313calc tmblck = 1
v1314com****************************************************************
v1315s.ni        (::0,0,0,3,0,1315,1327)
v1316com routine to clear the timed block flag
v1317com list=empty:empty:storage,time,ext,calc,incl,addr
v1318calc tmblck = 0
v1319com****************************************************************
v1320com
v1321com****************************************************************
v1322com
v1323com   program control
v1324com
v1325com****************************************************************
v1326com
v1327com****************************************************************
v1328s.perform   (name::3,28,9,6,0,1328,1335)
v1329com routine to invoke a procedure
v1330com list=procname:empty:storage,time,ext,calc,incl,addr
v1331begin stext
v1332           call    @<name>         ;perform procedure <name>
v1333endtext
v1334calc romptr = romptr + 3
v1335com****************************************************************
v1336s.jmpt      (val,loc:0,8:8,36,10,9,0,1336,1346)
v1337com routine to branch on true condition
v1338com list = value,location:precision:storage,time,ext,calc,incl,addr
v1339begin stext
v1340;branch on true
v1341           mov     AL,<val>        ;load value into accumulator
v1342           cmp     AL,0            ;compare to zero
v1343           jnz     <loc>           ;jump to <location> if true (=1)
v1344endtext
v1345calc romptr = romptr + 8
v1346com****************************************************************
v1347s.jmpf      (val,loc:0,8:8,36,10,9,0,1347,1357)
v1348com routine to branch on false condition
v1349com list = value,location:precision:storage,time,ext,calc,incl,addr
v1350begin stext
v1351;branch on false
v1352           mov     AL,<val>        ;load value into accumulator
```

```
v1353           cmp     AL,0              ;compare to zero
v1354           jz      <loc>             ;jump to <loc> if false(=0)
v1355endtext
v1356calc romptr = romptr + 8
v1357com***************************************************************
v1358s.loc       (loc::1,3,1,6,0,1358,1365)
v1359com routine to define a label
v1360com list=loc-name:empty;storage,time,ext,calc,incl,addr
v1361begin stext
v1362<loc>:      nop                      :define location <loc>
v1363endtext
v1364calc romptr = romptr + 1
v1365com***************************************************************
v1366s.every     (nam::6,43,12,9,0,1366,1376)
v1367com routine to define dummy function for every-period statment
v1368com list=proc-name:empty;storage,time,ext,calc,incl,addr
v1369begin stext
v1370;dummy procedure for every-period type contingency
v1371@<nam>:     nop                      ;dummy function entry point
v1372            mov     <nam>,1          ;force function value to true(1)
v1373            ret                      ;return to monitor
v1374endtext
v1375calc romptr = romptr + 6
v1376com***************************************************************
v1377s.forcons   (lwr,upr,slab,elab:0,16,0,16,0,65535:7,44,11,15,0,1377,1393)
v1378com routine to set-up loop with constant bounds
v1379com list=lower-bound,upper-bound,start-label,end-label:
v1380com      three precisions,max-loop-count:
v1381com      storage,time,ext,calc,incl,addr
v1382com max-loop-count = 65535
v1383begin stext
v1384;set-up for constant bounds loop
v1385           mov     AX,upr            ;move upper-bound to AX
v1386           sub     AX,lwr            ;calculate number of loop
v1387                                     ;iterations (upr - lwr)
v1388           xchg    CX                ;number of iterations to
v1389                                     ;CX (count) register
v1390<slab>:    nop                       ;start of loop
v1391endtext
v1392calc romptr = romptr + 7
v1393com***************************************************************
v1394s.forend    (slab,elab::2,17,2,8,0,1394,1403)
v1395com routine to end a for-loop
v1396com list=start-label,end-label:empty:storage,time,ext,calc,incl,addr
v1397begin stext
v1398;end a loop
v1399<elab>:    loop    <slab>            ;decrement CX register. if CX=0
v1400                                     ;exit loop, else return to <slab>
v1401endtext
v1402calc romptr = romptr + 2
v1403com***************************************************************
v1404s.whend     (whend,whtop::3,10,3,7,0,1404,1412)
```

```
v1405com primitive marking the end of a while do statement if executed
v1406com upon a true condition
v1407com list=end-label,top-label:storage,time,ext,calc,incl,addr
v1408begin stext
v1409<whend>:   jmp      <whtop>        ;jump to top of while
v1410endtext
v1411calc romptr = romptr + 3
v1412com****************************************************************
v1413s.whilecon  (arg1,whend,whtop:0,8:11,42,12,10,0,1413,1424)
v1414com primitive to generate a while do statement.  used with the
v1415com primitive whend to mark the end of the executable statments
v1416com list=logic expression,end-label,top-label:precision:storage,
v1417         time,ext,calc,incl,addr
v1418begin stext
v1419<whend>:   mov      AL,<arg1>      ;get result of logical exp
v1420           and      AL
v1421           jmp      <whend> + 3
v1422endtext
v1423calc romptr = romptr + 11
v1424com****************************************************************
v1425s.fixedwait (time:0,1000,0,10:10,43,10,4,17,1425,1443)
v1426com routine to delay a fixed period of time in increments of 5 usec
v1427com for the 8mhz 8086
v1428com list=time:delay,,resolution:storage,time,ext,calc,incl,addr
v1429calc cnt = time/5
v1430attr time = time
v1431if flt .eq. 1 skip 10
v1432begin stext
v1433;
v1434;wait <time> u-seconds (for 8MHz clock)
v1435;
v1436           mov      CX,<cnt>       ;load loop count and decrement
v1437           mov      AL,1           ;dummy instruction
v1438           cmp      AX,0000H       ;dummy instruction
v1439           loopnz   $ - 5          ;loop count until time is up
v1440endtext
v1441calc romptr = romptr + 10
v1442call s.fixedwait5
v1443com****************************************************************
v1444s.fixedwait5(time:0,1000,0,10:8,27,8,4,0,1444,1459)
v1445com routine to delay a fixed period of time in increments of 5 usec
v1446com for the 5mhz 8086 and 8087
v1447com list=time:delay,,resolution:storage,time,ext,calc,incl,addr
v1448calc cnt = time/5
v1449attr time = time
v1450begin stext
v1451;
v1452;wait <time> u-seconds (for 5 MHZ clock)
v1453;
v1454           mov      CX,<cnt>       ;load loop count and decrement
v1455           mov      AX,1           ;dummy instruction
v1456           loopnz   $ - 2          ;loop count until time is up
```

```
v1457endtext
v1458calc romptr = romptr + 8
v1459com****************************************************************
v1460s.assign    (var,data:0,8,0,8:6,26,8,9,0,1460,1470)
v1461com routine to assign a value of one variable to another variable
v1462com list=variable,data:variableprecision,dataprecision:
v1463com      storage,time,ext,calc,incl,addr
v1464begin stext
v1465;assign value of one variable to another variable (8-bit)
v1466            mov      AL,<data>      ;assign <data>
v1467            mov      <var>,AL       ;to <var>
v1468endtext
v1469calc romptr = romptr + 6
v1470com****************************************************************
v1471s.assign    (var,data:0,16,0,16:8,26,10,9,0,1471,1481)
v1472com routine to assign a value of one variable to another variable
v1473com list=variable,data:variableprecision,dataprecision:
v1474com      storage,time,ext,calc,incl,addr
v1475begin stext
v1476;assign value of one variable to another variable (16-bit)
v1477            mov      AX,<data>      ;assign <data>
v1478            mov      <var>,AX       ;to <var>
v1479endtext
v1480calc romptr = romptr + 8
v1481com****************************************************************
v1482s.assigncons(var,data:0,8,0,8:6,26,8,9,0,1482,1492)
v1483com routine to assign a value of a constant to a variable
v1484com list=variable,data:variableprecision,dataprecision:
v1485com      storage,time,ext,calc,incl,addr
v1486begin stext
v1487;assign value of a constant to a variable (8-bit)
v1488            mov      AL,<data>      ;assign <data>
v1489            mov      <var>,AL       ;to <var>
v1490endtext
v1491calc romptr = romptr + 6
v1492com****************************************************************
v1493s.assigncons(var,data:0,16,0,16:6,26,8,9,0,1493,1503)
v1494com routine to assign a value of a constant to a variable
v1495com list=variable,data:variableprecision,dataprecision:
v1496com      storage,time,ext,calc,incl,addr
v1497begin stext
v1498;assign value of a constant to a variable (16-bit)
v1499            mov      AL,<data>      ;assign <data>
v1500            mov      <var>,AL       ;to <var>
v1501endtext
v1502calc romptr = romptr + 6
v1503com****************************************************************
v1504s.fassign   (obj,src::0,0,0,0,0,1504,1510)
v1505com routine to assign value of a variable to another variable
v1506com list=variable,data:variableprecision,dataprecision:
v1507com      storage,time,ext,calc,incl,addr
v1508begin stext
```

```
v1509endtext
v1510com****************************************************************
v1511s.sensecond (signam:0,8,0,128:2,10,2,4,11,1511,1526)
v1512com routine to detect condition-type input
v1513com list=source:input lines,,conditions,,;storage,time,ext,calc
v1514com      incl,addr
v1515calc inport = inport + 1
v1516begin stext
v1517;detect contition-type input (8-bit)
v1518           in       AL,<inport>    ;sense environmental data
v1519           mov      <signam>,AL
v1520endtext
v1521calc romptr = romptr + 2
v1522incl h.sensecond (<signam>,<inport>:8)
v1523if latflg .ne. 1 skip 2
v1524incl h.latch (<signam>,<inport>:8)
v1525call s.resetlatch (<signam>,<inport>)
v1526com****************************************************************
v1527s.sensecond (signam:0,16,0,128:2,10,2,4,11,1527,1542)
v1528com routine to detect condition-type input
v1529com list=source:input lines,,conditions,,;storage,time,ext,calc
v1530com      incl,addr
v1531calc inport = inport + 1
v1532begin stext
v1533;detect condition-type input (16-bit)
v1534           in       AX,<inport>    ;sense environmental data
v1535           mov      <signam>,AX
v1536endtext
v1537calc romptr = romptr + 2
v1538incl h.sensecond (<signam>,<inport>:16)
v1539if latflg .ne. 1 skip 2
v1540incl h.latch (<signam>,<inport>:16)
v1541call s.resetlatch (<signam>,<inport>)
v1542com****************************************************************
v1543s.issuecond (signam:0,8,0,128:2,10,2,4,11,1543,1555)
v1544com routine to send condition-type output
v1545com list=sink:output-lines,,events,,:storage,time,ext,calc
v1546com      incl,addr
v1547calc outprt = outprt + 1
v1548begin stext
v1549;send condition-type output (8-bit)
v1550           mov      AL,<signam>    ;issue control
v1551           out      <outprt>,AL
v1552endtext
v1553calc romptr = romptr + 2
v1554incl h.issuecond (<signam>,<outprt>:8)
v1555com****************************************************************
v1556s.issuecond (signam:0,16,0,128:2,10,2,4,11,1556,1568)
v1557com routine to send condition-type output
v1558com list=sink:output-lines,,events,,:storage,time,ext,calc
v1559com      incl,addr
v1560calc outprt = outprt + 1
```

```
v1561begin stext
v1562;send condition-type output (16-bit)
v1563           mov      AX,<signam>     ;issue control
v1564           out      <outprt>,AX
v1565endtext
v1566calc romptr = romptr + 2
v1567incl h.issuecond (<signam>,<outprt>:16)
v1568com****************************************************************
v1569s.end       (::0,0,0,0,7,1569,1585)
v1570com routine to end software listing
v1571com list=empty:empty:storage,time,ext,calc,incl,addr
v1572begin stext
v1573           end                      ;software listing complete
v1574endtext
v1575com include sufficient eprom and static ram memory
v1576incl h.memory    (::)
v1577com****************************************************************
v1578com
v1579com****************************************************************
v1580com
v1581com  single precision integer arithmitic
v1582com
v1583com****************************************************************
v1584com
v1585com****************************************************************
v1586s.add       (rslt,arg1,arg2:0,8,0,8,0,8:9,41,12,10,0,1586,1597)
v1587com routine to add two 8-bit numbers
v1588com list=result,argument1,argument2:precisions:storage,time,ext,calc
v1589com      incl,addr
v1590begin stext
v1591;add 8-bit <arg1> + <arg2> = <rslt>
v1592           mov     AL,<arg1>        ;fetch first argument
v1593           add     AL,<arg2>        ;add second argument to first
v1594           mov     <rslt>,AL        ;store answer in <rslt>
v1595endtext
v1596calc romptr = romptr + 9
v1597com****************************************************************
v1598s.addck     (rslt,arg1,arg2:0,8,0,8,0,8:21,96,25,19,0,1598,1618)
v1599com routine to add two 8-bit numbers and return an 8-bit number
v1600com regardless of overflow or underflow.  on overflow, 7FH
v1601com is stored in <rslt>.  on underflow, -80H is stored in <rslt>.
v1602com list=result,argument1,argument2:precisions:storage,time,ext,calc
v1603com      incl,addr
v1604begin stext
v1605;add 8-bit with bounds check <arg1> + <arg2> = <rslt>
v1606           mov     AL,<arg1>        ;fetch first argument
v1607           add     AL,<arg2>        ;add second argument to first
v1608           jno     $ + 13           ;on no overflow store answer in
v1609                                    ;<rslt>
v1610           jb      $ + 7            ;if carry = 1, then underflow
v1611                                    ;and store largest negative value
v1612           mov     AL,7FH           ;else store largest positive value
```

```
v1613           jmp     $ + 5
v1614           mov     AL,-80H         ;largest negative number
v1615           mov     <rslt>,AL       ;store answer in <rslt>
v1616endtext
v1617calc romptr = romptr + 21
v1618com****************************************************************
v1619s.addck     (rslt,arg1,arg2:0,16,0,16,0,16:21,96,25,19,0,1619,1639)
v1620com routine to add two 16-bit numbers and return an 16-bit number
v1621com regardless of overflow or underflow.  on overflow, 7FFFH
v1622com is stored in <rslt>.  on underflow, -8000H is stored in <rslt>.
v1623com list=result,argument1,argument2:precisions:storage,time,ext,calc
v1624com      incl,addr
v1625begin stext
v1626;add 16-bit with bounds check <arg1> + <arg2> = <rslt>
v1627           mov     AX,<arg1>       ;fetch first argument
v1628           add     AX,<arg2>       ;add second argument to first
v1629           jno     $ + 13          ;on no overflow store answer in
v1630                                   ;<rslt>
v1631           jb      $ + 7           ;if carry = 1, then underflow
v1632                                   ;and store largest negative value
v1633           mov     AX,7FFFH        ;else store largest positive value
v1634           jmp     $ + 5
v1635           mov     AX,-8000        ;largest negative number
v1636           mov     <rslt>,AX       ;store answer in <rslt>
v1637endtext
v1638calc romptr = romptr + 21
v1639com****************************************************************
v1640s.add       (rslt,arg1,arg2:0,16,0,16,0,16:12,44,12,10,0,1640,1651)
v1641com routine to add two 16-bit numbers
v1642com list=result,argument1,argument2:precisions:storage,time,ext,calc
v1643com      incl,addr
v1644begin stext
v1645;add 16-bit <arg1> + <arg2> = <rslt>
v1646           mov     AX,<arg1>       ;fetch first argument
v1647           add     AX,<arg2>       ;add second argument to first
v1648           mov     <rslt>,AX       ;store answer in <rslt>
v1649endtext
v1650calc romptr = romptr + 12
v1651com****************************************************************
v1652s.sub       (rslt,arg1,arg2:0,8,0,8,0,8:9,38,3,10,0,1652,1663)
v1653com routine to subtract two 8-bit numbers
v1654com list=result,argument1,argument2:precisions:storage,time,ext,calc
v1655com      incl,addr
v1656begin stext
v1657;subtract 8-bit <arg1> - <arg2> = <rslt>
v1658           mov     AL,<arg1>       ;fetch subtrahend
v1659           sub     AL,<arg2>       ;fecth and subtract minuend
v1660           mov     <rslt>,AL       ;store answer in <rslt>
v1661endtext
v1662calc romptr = romptr + 9
v1663com****************************************************************
v1664s.sub       (rslt,arg1,arg2:0,16,0,16,0,16:10,38,3,10,0,1664,1675)
```

```
v1665com routine to subtract two 16-bit numbers
v1666com list=result,argument1,argument2:precisions:storage,time,ext,calc
v1667com      incl,addr
v1668begin stext
v1669;subtract 16-bit <arg1> - <arg2> = <rslt>
v1670          mov     AX,<arg1>        ;fetch subtrahend
v1671          sub     AX,<arg2>        ;fecth and subtract minuend
v1672          mov     <rslt>,AX        ;store answer in <rslt>
v1673endtext
v1674calc romptr = romptr + 10
v1675com****************************************************************
v1676s.mul        (rslt,arg1,arg2:0,8,0,8,0,8:17,114,3,11,0,1676,1688)
v1677com routine to multiply two 8-bit numbers
v1678com list=result,argument1,argument2:precisions:storage,time,ext,calc
v1679com      incl,addr
v1680begin stext
v1681;multiply 8-bit <arg1> * <arg2> = <rslt>
v1682          mov     AL,<arg1>        ;fetch multiplier
v1683          mov     CL,<arg2>        ;fetch multiplicand
v1684          mul     CL               ;multiply with answer in AX reg
v1685          mov     <rslt>,AX        ;store answer in <rslt>
v1686endtext
v1687calc romptr = romptr + 17
v1688com****************************************************************
v1689s.mul        (rsh,rsl,arg1,arg2:0,16,0,16,0,16,0,16:18,191,4,14,0,1689,1704)
v1690com routine to multiply two 16-bit unsigned numbers
v1691com list=result(upper),result(lower),argument1,argument2:precision:
v1692com      storage,time,ext,calc,incl,addrs
v1693begin stext
v1694;multiply 16-bit <arg1> * <arg2> = <rslt>
v1695          mov     AX,<arg1>        ;fetch multiplier
v1696          mov     CX,<arg2>        ;fetch multiplicand
v1697          mul     CX               ;multiply with answer in
v1698                                   ;upper word in DX reg
v1699                                   ;lower word in AX reg
v1700          mov     <rsh>,DX         ;store upper word in <rsh>
v1701          mov     <rsl>,AX         ;store lower word in <rsl>
v1702endtext
v1703calc romptr = romptr + 18
v1704com****************************************************************
v1705s.imul8      (rslt,arg1,arg2:0,8,0,8,0,8:11,140,14,11,0,1705,1717)
v1706com routine to multiply two 8-bit numbers without overflow
v1707com checking and returning an 8-bit result
v1708com list = result,argument1,argument2:precision:storage,time,ext,
v1709com        calc,incl,addrs
v1710begin stext
v1711          mov     AL,<arg1>        ;fetch multiplier
v1712          mov     DL,<arg2>        ;fetch multiplicand
v1713          imul    DL
v1714          mov     <rslt>,AL        ;store <rslt>
v1715endtext
v1716calc romptr = romptr + 11
```

```
v1717com****************************************************************
v1718s.imulom8   (rslt,arg1,arg2:0,8,0,8,0,8:29,207,35,23,0,1718,1742)
v1719com routine to multiply two 8-bit numbers with overflow
v1720com resulting in maximum positive or negative 8-bit result
v1721com list = result,argument1,argument2:precision:storage,time,ext,
v1722com        calc,incl,addrs
v1723begin stext
v1724           mov     AL,<arg1>       ;fetch multiplier
v1725           mov     DL,<arg2>       ;fetch mulitplicand
v1726           imul    DL
v1727           cmp     AH,80H          ;check if result is positive
v1728                                   ;or negative
v1729           jz      immel           ;if zero result is negative
v1730           cmp     AH,00H          ;check for significant digits
v1731                                   ;in upper byte of positive
v1732           jz      imome           ;if zero, store <rslt>
v1733           mov     AL,7FH          ;else stuff in largest positive
v1734           jmp     imome
v1735           cmp     AH,00H          ;check for significant digits
v1736                                   ;in upper byte of negative
v1737           jz      imome           ;if zero, store <rslt>
v1738immel:     mov     AL,80H          ;else stuff in largest negative
v1739imome:     mov     <rslt>,AL
v1740endtext
v1741calc romptr = romptr + 29
v1742com****************************************************************
v1743s.imulu8    (rslt,arg1,arg2:0,8,0,8,0,8:11,140,14,11,0,1743,1755)
v1744com routine to multiply two 8-bit numbers returning the upper
v1745com 8 bits of the product
v1746com list = result,argument1,argument2:precision:storage,time,ext,
v1747com        calc,incl,addrs
v1748begin stext
v1749           mov     AL,<arg1>       ;fetch multiplier
v1750           mov     DL,<arg2>       ;fetch multiplicand
v1751           imul    DL
v1752           mov     <rslt>,AH       ;store <rslt>
v1753endtext
v1754calc romptr = romptr + 11
v1755com****************************************************************
v1756s.imulex8   (rslt,arg1,arg2:0,16,0,8,0,8:24,183,29,19,0,1756,1776)
v1757com routine to multiply two 8-bit numbers returning a
v1758com 16 bit product
v1759com list = result,argument1,argument2:precision:storage,time,ext,
v1760com        calc,incl,addrs
v1761begin stext
v1762           mov     AL,<arg1>       ;fetch multiplier
v1763           cbw                     ;sign extend the multiplier
v1764           mov     DX,AX
v1765           mov     AL,<arg2>       ;fetch multiplicand
v1766           cbw                     ;sign extend multiplicand
v1767           imul    DX
v1768           jnc     im18            ;check to see if upper byte
```

```
v1769           jno     im18            ;has significant digits
v1770           jmp     im28
v1771im18:      mov     BX,AX           ;store lower 8 bits
v1772           xor     AX,AX           ;clear upper 8 bits
v1773im28:      mov     <rslt>,AX       ;store <rslt>
v1774endtext
v1775calc romptr = romptr + 24
v1776com***************************************************************
v1777s.imul16    (rsh,rsl,arg1,arg2:0,16,0,16,0,16:14,156,14,11,0,1777,1789)
v1778com routine to multiply two 16-bit numbers without overflow
v1779com list = result,argument1,argument2:precision:storage,time,ext,
v1780com        calc,incl,addrs
v1781begin stext
v1782           mov     AX,<arg1>       ;fetch multiplicand
v1783           mov     DX,<arg2>       ;fetch multiplier
v1784           imul    DX
v1785           mov     <rsl>,AX
v1786           mov     <rsh>,DX
v1787endtext
v1788calc romptr = romptr + 14
v1789com***************************************************************
v1790s.imulom16  (rslt,arg1,arg2:0,16,0,16,0,16:54,290,54,23,0,1790,1814)
v1791com routine to multiply two 16-bit numbers with overflow
v1792com returning maximum negative or positive 16-bit result
v1793com list = result,argument1,argument2:precision:storage,time,ext,
v1794com        calc,incl,addrs
v1795begin stext
v1796           mov     AX,<arg1>       ;fetch multiplier
v1797           mov     CX,<arg2>       ;fetch multiplicand
v1798           imul    CX
v1799           cmp     DX,8000H        ;check if result positive
v1800           jz      imm16           ;or negative
v1801           cmp     DX,0000H        ;check for significant
v1802           jz      imom6           ;digits in upper byte
v1803           mov     AX,FFFFH        ;stuff in largest positive
v1804           mov     DX,7FFFH
v1805           jmp     imom6
v1806           cmp     DX,0000H
v1807           jz      imom6
v1808imm16:     mov     AX,0000H
v1809           mov     DX,7FFFH
v1810imom6:     mov     <rsh>,DX
v1811           mov     <rsl>,AX
v1812endtext
v1813calc romptr = romptr + 54
v1814com***************************************************************
v1815s.imulu16   (rslt,arg1,arg2:0,16,0,16,0,16:11,141,11,11,0,1815,1827)
v1816com routine to multiply two 16-bit numbers returning the upper
v1817com 16 bits of the product
v1818com list = result,argument1,argument2:precision:storage,time,ext,
v1819com        calc,incl,addrs
v1820begin stext
```

```
v1821           mov     AX,<arg1>       ;fetch multiplicand
v1822           mov     DX,<arg2>       ;fetch multiplier
v1823           imul    DX
v1824           mov     <rslt>,DX
v1825endtext
v1826calc romptr = romptr + 11
v1827com****************************************************************
v1828s.div       (rslt,arg1,arg2:0,8,0,8,0,8:11,136,11,11,0,1828,1840)
v1829com routine to divide two 8-bit numbers
v1830com list = result,argument1,argument2:precision:storage,time,ext,
v1831com        calc,incl,addrs
v1832begin stext
v1833;divide 8-bit <arg1> / <arg2> = <rslt>
v1834           mov     AL,<arg1>       ;fetch dividend
v1835           mov     CL,<arg2>       ;fetch divisor
v1836           div     CL              ;divide with answer in AL reg
v1837           mov     <rslt>,AL       ;store answer in <rslt>
v1838endtext
v1839calc romptr = romptr + 11
v1840com****************************************************************
v1841s.div       (rslt,arg1,arg2:0,16,0,16,0,16:13,208,13,11,0,1841,1861)
v1842com routine to divide two 16-bit numbers
v1843com list = result,argument1,argument2:precision:storage,time,ext,
v1844com        calc,incl,addrs
v1845begin stext
v1846;divide 16-bit <arg1> / <arg2> = <rslt>
v1847           mov     AX,<arg1>       ;fetch dividend
v1848           mov     CX,<arg2>       ;fetch divisor
v1849           div     CX              ;divide with answer in AX reg
v1850           mov     <rslt>,AX       ;store answer in <rslt>
v1851endtext
v1852calc romptr = romptr + 13
v1853com****************************************************************
v1854com
v1855com****************************************************************
v1856com
v1857com  floating point arithmetic operations
v1858com
v1859com****************************************************************
v1860com
v1861com****************************************************************
v1862s.8087cw    (::3,20,3,11,0,1862,1874)
v1863com routine to load the 8087 NDP control word prior to any NDP
v1864com floating point calculation.  control word specifies short real
v1865com calculations, rounding to nearest or even with affine
v1866com infinity control.
v1867com list=empty:empty:storage,time,ext,calc,incl,addrs
v1868begin stext
v1869           org     <romptr>
v1870cntwd:     dw      00bfH           ;defines control word
v1871           fldcw   cntwd           ;load control word into NDP
v1872endtext
```

```
v1873calc cword = cword + 1
v1874com***************************************************************
v1875s.funpack    (arg1,arg2:0,24,0,24:44,136,59,45,0,1875,1921)
v1876com routine to unpack floating point operands and store them
v1877com in pre-defined ram storage. format of arg1 and arg2:
v1878com lower mantissa, middle mantissa, high mantissa + part of exp,
v1879com exponent + mantissa sign
v1880com list = argument1,argument2:precision:storage,time,ext,calc,
v1881com        incl,addrs
v1882begin stext
v1883;unpack <arg1>
v1884           mov     sign1,0000H       ;presuppose sign is positive
v1885           mov     AX,<arg1> + 2     ;move exp/mantissa into AX
v1886           and     AX,007FH          ;mask out sign and exponent
v1887           stc                       ;set carry to shift in
v1888           rcr     AX                ;invisible bit
v1889           mov     uman1,AX          ;store in uman1
v1890           mov     AX,<arg1>         ;get lower mantissa
v1891           rcr     AX                ;rotate low order bit of umant
v1892           mov     lman1,AX          ;into high order bit of lmant
v1893                                     ;and store result in lman1
v1894           mov     AX,<arg1> + 2
v1895           cmp     AX,0F000H         ;check for sign of mantissa
v1896           js      $ + 5
v1897           mov     sign1,0001H       ;sign is negative
v1898           mov     AX,<arg1> + 2     ;mask exponent and store
v1899           and     AX,7F80H
v1900           mov     exp1,AX
v1901;unpack <arg2>
v1902           mov     sign2,0000H       ;presuppose sign is positive
v1903           mov     AX,<arg2> + 2     ;move exp/mantissa into AX
v1904           and     AX,007FH          ;mask out sign and exponent
v1905           stc                       ;set carry to shift in
v1906           rcr     AX                ;invisible bit
v1907           mov     uman2,AX          ;store in uman1
v1908           mov     AX,<arg2>         ;get lower mantissa
v1909           rcr     AX                ;rotate low order bit of umant
v1910           mov     lman2,AX          ;into high order bit of lmant
v1911                                     ;and store result in lman1
v1912           mov     AX,<arg2> + 2
v1913           cmp     AX,0F000H         ;check for sign of mantissa
v1914           js      $ + 5
v1915           mov     sign2,0001H       ;sign is negative
v1916           mov     AX,<arg2> + 2     ;mask exponent and store
v1917           and     AX,7F80H
v1918           mov     exp2,AX
v1919endtext
v1920calc romptr = romptr + 44
v1921com***************************************************************
v1922s.fpack      (rslt:0,16:32,101,44,21,0,1922,1944)
v1923com routine to pack an unpacked arithmetic result of a floating
v1924com point operation.  format: upacked version to intel short form
```

```
v1925com arranged into two 16-bit words.
v1926com list = result:precision:storage,time,ext,calc,
v1927com        incl,addrs
v1928begin stext
v1929;pack <rslt>
v1930           mov     AX,lmant3
v1931           sal     AX                  ;shift msb of lmant into carry
v1932           mov     <rslt>,AX           ;store lmant in <rslt>
v1933           mov     DX,umant3
v1934           rcl     DX                  ;shift msb of lmant into lsb of
v1935           and     DX,7FH              ;of umant and mask out overflow
v1936           or      AX,DX               ;pack umant and lmant
v1937           mov     DX,exp3
v1938           or      AX,DX               ;pack mantissa and exponent
v1939           mov     DX,sign3
v1940           or      AX,DX               ;pack sign
v1941           mov     <rslt>+2,AX
v1942endtext
v1943calc romptr = romptr + 20
v1944com****************************************************************
v1945s.fmul      (rslt,arg1,arg2:0,24,0,24,0,24:2,2,2,15,6,1945,1961)
v1946com routine to perform a short real floating point multiply
v1947com without the 8087 ndp
v1948com list = result,argument1,argument2:precision:storage,time,ext,
v1949com        calc,incl,addrs
v1950if flt .eq. 0 skip 2
v1951call s.nfmul     (<rslt>,<arg1>,<arg2>:)
v1952skip 8
v1953call s.funpack   (<arg1>,<arg2>:)
v1954begin stext
v1955;fp multiply <arg1> * <arg2> = <rslt>
v1956        (fp multiply <rslt> = <arg1> * <arg2>)
v1957;
v1958endtext
v1959call s.fpack     (<rslt>::)
v1960calc romptr = romptr + 2
v1961com****************************************************************
v1962s.nfmul     (rslt,arg1,arg2:0,24,0,24,0,24:11,291,17,14,5,1962,1977)
v1963com routine to perform a floating point multiply with 8087 NDP
v1964com list = result,argument1,argument2:precision:storage,time,ext,
v1965com        calc,incl,addrs
v1966if cword .gt. 0 skip 1
v1967call s.8087cw    (::)
v1968begin stext
v1969;nfp multiply <arg1> * <arg2> = <rslt>
v1970           fld     <arg1>              ;load <arg1> onto ndp stack(0)
v1971           fmul    <arg2>              ;load <arg2> onto ndp stack(1)
v1972                                       ;perform multiply, <rslt> to st(0)
v1973           fwait                       ;cpu wait for fp multiply
v1974           fstp    <rslt>              ;store <rslt> and pop ndp stack
v1975endtext
v1976calc romptr = romptr + 11
```

```
v1977com***************************************************************
v1978s.fsub      (rslt,arg1,arg2:0,24,0,24,0,24:0,0,0,14,6,1978,1993)
v1979com routine to perform a short real floating point subtraction
v1980com without the  8087 ndp
v1981com list = result,argument1,argument2:precision:storage,time,ext,
v1982com        calc,incl,addrs
v1983if flt .eq. 0 skip 2
v1984call s.nfsub     (<rslt>,<arg1>,<arg2>:)
v1985skip 7
v1986call s.funpack   (<arg1>,<arg2>:)
v1987begin stext
v1988;fp subtract <arg1> - <arg2> = <rslt>
v1989        (fp sub <rslt> = <arg1> - <arg2>)
v1990;
v1991endtext
v1992calc romptr = romptr + 0
v1993com***************************************************************
v1994s.nfsub     (rslt,arg1,arg2:0,24,0,24,0,24:11,286,17,16,6,1994,2011)
v1995com routine to perform a short real floating point subtraction
v1996com with an 8087 ndp
v1997com list = result,argument1,argument2:precision:storage,time,ext,
v1998com        calc,incl,addrs
v1999if cword .gt. 0 skip 1
v2000call s.8087cw    (::)
v2001begin stext
v2002;nfp subtract <arg1> - <arg2> = <rslt>
v2003           fld     <arg1>          ;load <arg1> onto ndp stack (0)
v2004           fsub    <arg2>          ;fetch <arg2> and subtract with
v2005                                   ;<rslt> in stack (0)
v2006           fwait                   ;cpu wait for subtract
v2007           fstp    <rslt>          ;store <rslt> and pop ndp stack
v2008;
v2009endtext
v2010calc romptr = romptr + 11
v2011com***************************************************************
v2012s.fadd      (rslt,arg1,arg2:0,24,0,24,0,24:0,0,0,14,6,2012,2027)
v2013com routine to perform a short real floating point addition
v2014com without the 8087 ndp
v2015com list = result,argument1,argument2:precision:storage,time,ext,
v2016com        calc,incl,addrs
v2017if flt .eq. 0 skip 2
v2018call s.nfadd     (<rslt>,<arg1>,<arg2>:)
v2019skip 7
v2020call s.funpack   (<arg1>,<arg2>:)
v2021begin stext
v2022;fp add <arg1> + <arg2> = <rslt>
v2023        (fp add <rslt> = <arg1> + <arg2>)
v2024;
v2025endtext
v2026calc romptr = romptr + 0
v2027com***************************************************************
v2028s.nfadd      (rslt,arg1,arg2:0,24,0,24,0,24:11,286,17,16,6,2028,2045)
```

```
v2029com routine to perform a short real floating point addition
v2030com with the 8087 ndp
v2031com list = result,argument1,argument2:precision:storage,time,ext,
v2032com        calc,incl,addrs
v2033if cword .gt. 0 skip 1
v2034call s.8087cw    (::)
v2035begin stext
v2036;nfp add <arg1> + <arg2> = <rslt>
v2037           fld     <arg1>          ;load <arg1> onto ndp stack (0)
v2038           fadd    <arg2>          ;fetch <arg2> perform addition
v2039                                   ;with <rslt> to stack (0)
v2040           fwait                   ;cpu wait for addition
v2041           fstp    <rslt>          ;store <rslt> and pop ndp stack
v2042;
v2043endtext
v2044calc romptr = romptr + 11
v2045com*****************************************************************
v2046s.fdiv      (rslt,arg1,arg2:0,32,0,32,0,32:0,0,0,14,6,2046,2061)
v2047com routine to perform short real floating point divide without
v2048com the 8087 ndp
v2049com list = result,argument1,argument2:precision:storage,time,ext,
v2050com        calc,incl,addrs
v2051if flt .eq. 0 skip 2
v2052call s.nfdiv     (<rslt>,<arg1>,<arg2>:)
v2053skip 7
v2054call s.funpack   (<arg1>,<arg2>:)
v2055begin stext
v2056;fp div <arg1> / <arg2> = <rslt>
v2057        (fp div <rslt> = <arg1> / <arg2>)
v2058;
v2059endtext
v2060calc romptr = romptr + 0
v2061com*****************************************************************
v2062s.nfdiv     (rslt,arg1,arg2:0,32,0,32,0,32:11,391,17,16,6,2062,2079)
v2063com routine to perform a short real floating point division
v2064com with the 8087 ndp
v2065com list = result,argument1,argument2:precision:storage,time,ext,
v2066com        calc,incl,addrs
v2067if cword .gt. 0 skip 1
v2068call s.8087cw    (::)
v2069begin stext
v2070;nfp div <arg1> / <arg2> = <rslt>
v2071           fld     <arg1>          ;load <arg1> onto ndp stack(0)
v2072           fdiv    <arg2>          ;fetch <arg2> and perform division
v2073                                   ;with <rslt> to stack(0)
v2074           fwait                   ;cpu wait for divide
v2075           fstp    <rslt>          ;store <rslt> and pop ndp stack
v2076;
v2077endtext
v2078calc romptr = romptr + 11
v2079com*****************************************************************
v2080s.float     (rslt,arg:0,8,0,8:7,158,9,12,5,2080,2093)
```

```
v2081com routine to perform an integer to floating point conversion
v2082com list = result,argument1:precision:storage,time,ext,
v2083com        calc,incl,addrs
v2084if cword .gt. 0 skip 1
v2085call s.8087cw    (::)
v2086begin stext
v2087           org     <romptr>
v2088           fild    <arg>              ;load integer argument
v2089           fstp    <rslt>             ;store floating point result
v2090           fwait
v2091endtext
v2092calc romptr = romptr + 8
v2093com***************************************************************
v2094s.fix       (rslt,arg:0,16,0,16:8,158,9,10,0,2094,2105)
v2095com routine to convert a floating point value to an integer value
v2096com list = result,argument1:precision:storage,time,ext,
v2097com        calc,incl,addrs
v2098begin stext
v2099           org     <romptr>
v2100           fld     <arg>              ;load floating point value
v2101           fistp   <rslt>             ;store integer result
v2102           fwait
v2103endtext
v2104calc romptr = romptr + 8
v2105com***************************************************************
v2106s.fcons     (name,lmant,mmant,hmant,exp::0,0,0,10,0,2106,2117)
v2107com list = result,argument1,argument2:precision:storage,time,ext,
v2108com        calc,incl,addrs
v2109begin stext
v2110;fp constant assignment
v2111<name>:    db      <lmant>
v2112           db      <mmant>
v2113           db      <hmant>
v2114           db      <exp>
v2115endtext
v2116calc romptr = romptr + 0
v2117com***************************************************************
v2118s.nopck     (rslt:0,32:34,130,41,24,0,2118,2151)
v2119com routine to handle overflow and underflow conditions resulting
v2120com from 8087 ndp operations
v2121com list = result:precision:storage,time,ext,calc,
v2122com        incl,addrs
v2123begin stext
v2124           org     <romptr>
v2125stsw:      dw      0
v2126nopck:     fstsw   stsw               ;store status word in memory
v2127           mov     AX,stsw
v2128           cmp     AX,0010H           ;check for underflow condition
v2129           jz      undflw             ;underflow has occured
v2130           cmp     AX,0080H           ;check for overflow condition
v2131           jz      ovrflw             ;overflow has occured
v2132           jmp     endpck
```

```
v2133undflw:    mov     <rslt>,0FFFFH   ;stuff largest negative number
v2134                                   ;in <rslt>
v2135           mov     <rslt>+2,8070H
v2136           jmp     endpck
v2137ovrflw:    mov     <rslt>,0FFFFH   ;stuff largest positive number
v2138                                   ;in <rslt>
v2139           mov     <rslt>+2,0BFFFH
v2140endpck:    nop
v2141endtext
v2142calc romptr = romptr + 34
v2143com****************************************************************
v2144com
v2145com****************************************************************
v2146com
v2147com  logical operations
v2148com
v2149com****************************************************************
v2150com
v2151com****************************************************************
v2152s.and      (rslt,arg1,arg2:0,8,0,8,0,8:10,41,11,10,0,2152,2163)
v2153com routine to perform 8-bit logical and
v2154com list=result,argument1,argument2:precisions:storage,time,
v2155com      ext,calc,incl,addr
v2156begin stext
v2157;logical and (8-bit) <arg1> .and. <arg2> = <rslt>
v2158           mov     AL,<arg1>
v2159           and     AL,<arg2>
v2160           mov     <rslt>,AL
v2161endtext
v2162calc romptr = romptr + 10
v2163com****************************************************************
v2164s.and      (rslt,arg1,arg2:0,16,0,16,0,16:12,41,11,10,0,2164,2175)
v2165com routine to perform 16-bit logical and
v2166com list=result,argument1,argument2:precisions:storage,time,
v2167com      ext,calc,incl,addr
v2168begin stext
v2169;logical and. (16-bit) <arg1> .and. <arg2> = <rslt>
v2170           mov     AX,<arg1>
v2171           and     AX,<arg2>
v2172           mov     <rslt>,AX
v2173endtext
v2174calc romptr = romptr + 12
v2175com****************************************************************
v2176s.or       (rslt,arg1,arg2:0,8,0,8,0,8:10,41,12,10,0,2176,2187)
v2177com routine to perform 8-bit logical or
v2178com list=result,argument1,argument2:precisions:storage,time,
v2179com      ext,calc,incl,addr
v2180begin stext
v2181;logical or. (8-bit) <arg1> .or. <arg2>
v2182           mov     AL,<arg1>
v2183           or      AL,<arg2>
v2184           mov     <rslt>,AL
```

```
v2185endtext
v2186calc romptr = romptr + 10
v2187com****************************************************************
v2188s.or        (rslt,arg1,arg2:0,16,0,16,0,16:12,41,12,10,0,2188,2199)
v2189com routine to perform 16-bit logical or
v2190com list=result,argument1,argument2:precisions:storage,time,
v2191com      ext,calc,incl,addr
v2192begin stext
v2193;logical or. (16-bit) <arg1> .or. <arg2>
v2194           mov     AX,<arg1>
v2195           or      AX,<arg2>
v2196           mov     <rslt>,AX
v2197endtext
v2198calc romptr = romptr + 12
v2199com****************************************************************
v2200s.not       (rslt,arg:0,8,0,8:8,29,10,10,0,2200,2211)
v2201com routine to perform 8-bit logical not
v2202com list=result,argument1,argument2:precisions:storage,time,
v2203com      ext,calc,incl,addr
v2204begin stext
v2205;logical not. (8-bit) <rslt> = .not. <arg1>
v2206           mov     AL,<arg>
v2207           not
v2208           mov     <rslt>,AL
v2209endtext
v2210calc romptr = romptr + 8
v2211com****************************************************************
v2212s.not       (rslt,arg:0,16,0,16,0,16:10,29,10,10,0,2212,2223)
v2213com routine to perform 16-bit logical not
v2214com list=result,argument1,argument2:precisions:storage,time,
v2215com      ext,calc,incl,addr
v2216begin stext
v2217;logical not. (16-bit) <rslt> = .not. <arg1>
v2218           mov     AX,<arg>
v2219           not
v2220           mov     <rslt>,AX
v2221endtext
v2222calc romptr = romptr + 10
v2223com****************************************************************
v2224s.xor       (rslt,arg1,arg2:0,8,0,8:10,41,12,10,0,2224,2235)
v2225com routine to perform 8-bit logical exclusive or
v2226com list=result,argument1,argument2:precisions:storage,time,
v2227com      ext,calc,incl,addr
v2228begin stext
v2229;logical xor. (8-bit) <arg1> .xor. <arg2> = <rslt>
v2230           mov     AL,<arg1>
v2231           xor     AL,<arg2>
v2232           mov     <rslt>,AL
v2233endtext
v2234calc romptr = romptr + 10
v2235com****************************************************************
v2236s.xor       (rslt,arg1,arg2:0,16,0,16,0,16:10,41,12,10,0,2236,2247)
```

```
v2237com routine to perform 16-bit logical exclusive or
v2238com list=result,argument1,argument2:precisions:storage,time,
v2239com      ext,calc,incl,addr
v2240begin stext
v2241;logical xor. (16-bit) <arg1> .xor. <arg2> = <rslt>
v2242           mov     AX,<arg1>
v2243           xor     AX,<arg2>
v2244           mov     <rslt>,AX
v2245endtext
v2246calc romptr = romptr + 10
v2247com***************************************************************
v2248s.eq         (rslt,arg1,arg2:0,8,0,8,0,8:15,79,21,13,0,2248,2262)
v2249com routine to set result to true (1) if arg1 equals arg2 else
v2250com result is false (0)
v2251com list=result,argument1,argument2:precisions:storage,time,
v2252com      ext,calc,incl,addr
v2253begin stext
v2254;test for equality between <arg1> and <arg2> (8-bit)
v2255           mov     <rslt>,1         ;presuppose equality
v2256           mov     AL,<arg1>        ;fetch <arg1>
v2257           mov     BL,<arg2>        ;fetch <arg2>
v2258           jz      $+4              ;end routine if true
v2259           mov     <rslt>,0         ;not equal, <rslt> = 0
v2260endtext
v2261calc romptr = romptr + 15
v2262com***************************************************************
v2263s.eq         (rslt,arg1,arg2:0,16,0,16,0,16:18,79,21,13,0,2263,2277)
v2264com routine to set result to true (1) if arg1 equals arg2 else
v2265com result is false (0)
v2266com list=result,argument1,argument2:precisions:storage,time,
v2267com      ext,calc,incl,addr
v2268begin stext
v2269;test for equality between <arg1> and <arg2> (16-bit)
v2270           mov     <rslt>,1         ;presuppose equality
v2271           mov     AX,<arg1>        ;fetch <arg1>
v2272           cmp     AX,<arg2>        ;compare arguments
v2273           jz      $+4              ;end routine if true
v2274           mov     <rslt>,0         ;not equal, <rslt> = 0
v2275endtext
v2276calc romptr = romptr + 18
v2277com***************************************************************
v2278s.lt         (rslt,arg1,arg2:0,8,0,8,0,8:15,79,21,13,0,2278,2292)
v2279com routine to set result to true (1) if arg1 is less than arg2
v2280com else result is false (0)
v2281com list=result,argument1,argument2:precisions:storage,time,
v2282com      ext,calc,incl,addr
v2283begin stext
v2284;test if <arg1> less than <arg2> then <rslt> = 1 (8-bit)
v2285           mov     <rslt>,1         ;presuppose arg1 < arg2
v2286           mov     AL,<arg1>        ;fetch <arg1>
v2287           cmp     AL,<arg2>        ;compare arguments
v2288           js      $+4              ;end routine if true
```

```
v2289           mov     <rslt>,0        ;not < , <rslt> = 0
v2290endtext
v2291calc romptr = romptr + 15
v2292com****************************************************************
v2293s.lt        (rslt,arg1,arg2:0,16,0,16,0,16:18,79,21,13,0,2293,2307)
v2294com routine to set result to true (1) if arg1 is less than arg2
v2295com else result is false (0)
v2296com list=result,argument1,argument2:precisions:storage,time,
v2297com      ext,calc,incl,addr
v2298begin stext
v2299;test if <arg1> less than <arg2> then <rslt> = 1 (16-bit)
v2300           mov     <rslt>,1        ;presuppose arg1 < arg2
v2301           mov     AX,<arg1>       ;fetch <arg1>
v2302           cmp     AX,<arg2>       ;compare arguments
v2303           js      $+4             ;end routine if true
v2304           mov     <rslt>,0        ;not < , <rslt> = 0
v2305endtext
v2306calc romptr = romptr + 18
v2307com****************************************************************
v2308s.le        (rslt,arg1,arg2:0,8,0,8,0,8:15,79,21,13,0,2308,2322)
v2309com routine to set result to true (1) if arg1 is less than or equal
v2310com to arg2 else result is false (0)
v2311com list=result,argument1,argument2:precisions:storage,time,
v2312com      ext,calc,incl,addr
v2313begin stext
v2314;test if <arg1> less than or equal <arg2> then <rslt> = 1 (8-bit)
v2315           mov     <rslt>,1        ;presuppose arg1 < = arg2
v2316           mov     AL,<arg1>       ;fetch <arg1>
v2317           cmp     AL,<arg2>       ;compare arguments
v2318           jbe     $+4             ;end routine if true
v2319           mov     <rslt>,0        ;not < = , <rslt> = 0
v2320endtext
v2321calc romptr = romptr + 15
v2322com****************************************************************
v2323s.le        (rslt,arg1,arg2:0,16,0,16,0,16:18,79,21,13,0,2323,2337)
v2324com routine to set result to true (1) if arg1 is less than or equal
v2325com to arg2 else result is false (0)
v2326com list=result,argument1,argument2:precisions:storage,time,
v2327com      ext,calc,incl,addr
v2328begin stext
v2329;test if <arg1>less than <arg2> then<rslt> = 1 (16-bit)
v2330           mov     <rslt>,1        ;presuppose arg1 < = arg2
v2331           mov     AX,<arg1>       ;fetch <arg1>
v2332           cmp     AX,<arg2>       ;compare arguments
v2333           jbe     $+4             ;end routine if true
v2334           mov     <rslt>,0        ;not < = , <rslt> = 0
v2335endtext
v2336calc romptr = romptr + 18
v2337com****************************************************************
v2338s.gt        (rslt,arg1,arg2:0,8,0,8,0,8:15,79,21,13,0,2338,2352)
v2339com routine to set result to true (1) if arg1 is greater than arg2
v2340com else result is false (0)
```

```
v2341com list=result,argument1,argument2:precisions:storage,time,
v2342com      ext,calc,incl,addr
v2343begin stext
v2344;test if <arg1> greater than <arg2> then <rslt> = 1 (8-bit)
v2345           mov     <rslt>,1        ;presuppose arg1 > arg2
v2346           mov     AL,<arg1>       ;fetch <arg1>
v2347           cmp     AL,<arg2>       ;compare arguments
v2348           jg      $+4             ;end routine if true
v2349           mov     <rslt>,0        ;not > , <rslt> = 0
v2350endtext
v2351calc romptr = romptr + 15
v2352com****************************************************************
v2353s.gt        (rslt,arg1,arg2:0,16,0,16,0,16:18,79,21,13,0,2353,2367)
v2354com routine to set result to true (1) if arg1 is greater than arg2
v2355com else result is false (0)
v2356com list=result,argument1,argument2:precisions:storage,time,
v2357com      ext,calc,incl,addr
v2358begin stext
v2359;test if <arg1>greater than <arg2> then <rslt> = 1 (16-bit)
v2360           mov     <rslt>,1        ;presuppose arg1 > arg2
v2361           mov     AX,<arg1>       ;fetch <arg1>
v2362           cmp     AX,<arg2>       ;compare arguments
v2363           jg      $+4             ;end routine if true
v2364           mov     <rslt>,0        ;not > , <rslt> = 0
v2365endtext
v2366calc romptr = romptr + 18
v2367com****************************************************************
v2368s.ge        (rslt,arg1,arg2:0,8,0,8,0,8:15,79,21,13,0,2368,2382)
v2369com routine to set result to true (1) if arg1 is greater than or
v2370com equal to arg2 else result is false (0)
v2371com list=result,argument1,argument2:precisions:storage,time,
v2372com      ext,calc,incl,addr
v2373begin stext
v2374;test if <arg1> greater or equal <arg2> then <rslt> = 1 (8-bit)
v2375           mov     <rslt>,1        ;presuppose arg1 > = arg2
v2376           mov     AL,<arg1>       ;fetch <arg1>
v2377           cmp     AL,<arg2>       ;compare arguments
v2378           jge     $+4             ;end routine if true
v2379           mov     <rslt>,0        ;not > = , <rslt> = 0
v2380endtext
v2381calc romptr = romptr + 15
v2382com****************************************************************
v2383s.ge        (rslt,arg1,arg2:0,16,0,16,0,16:18,79,21,13,0,2383,2397)
v2384com routine to set result to true (1) if arg1 is greater than or
v2385com equal to arg2 else result is false (0)
v2386com list=result,argument1,argument2:precisions:storage,time,
v2387com      ext,calc,incl,addr
v2388begin stext
v2389;test if <arg1> greater or equal<arg2> then <rslt> = 1 (16-bit)
v2390           mov     <rslt>,1        ;presuppose arg1 > = arg2
v2391           mov     AX,<arg1>       ;fetch <arg1>
v2392           cmp     AX,<arg2>       ;compare arguments
```

```
v2393           jge     $+4               ;end routine if true
v2394           mov     <rslt>,0          ;not > = , <rslt> = 0
v2395endtext
v2396calc romptr = romptr + 18
v2397com****************************************************************
v2398s.ne        (rslt,arg1,arg2:0,8,0,8,0,8:15,79,21,13,0,2398,2412)
v2399com routine to set result to true (1) if arg1 is not equal to arg2
v2400com else result is false (0)
v2401com list=result,argument1,argument2:precisions:storage,time,
v2402com      ext,calc,incl,addr
v2403begin stext
v2404;test if <arg1> not equal <arg2> then <rslt> = 1 (8-bit)
v2405          mov     <rslt>,1          ;presuppose inequality
v2406           mov     AL,<arg1>         ;fetch <arg1>
v2407           cmp     AL,<arg2>         ;compare arguments
v2408           jne     $+4               ;end routine if true
v2409           mov     <rslt>,0          ;equal, <rslt> = 0
v2410endtext
v2411calc romptr = romptr + 15
v2412com****************************************************************
v2413s.ne        (rslt,arg1,arg2:0,16,0,16,0,16:18,79,21,13,0,2413,2435)
v2414com routine to set result to true (1) if arg1 is not equal to arg2
v2415com else result is false (0)
v2416com list=result,argument1,argument2:precisions:storage,time,
v2417com      ext,calc,incl,addr
v2418begin stext
v2419;test if <arg1> not equal <arg2> then <rslt> = 1 (16-bit)
v2420           mov     <rslt>,1          ;presuppose inequality
v2421           mov     AX,<arg1>         ;fetch <arg1>
v2422           cmp     AX,<arg2>         ;compare arguments
v2423           jne     $+4               ;end routine if true
v2424           mov     <rslt>,0          ;equal, <rslt> = 0
v2425endtext
v2426calc romptr = romptr + 18
v2427com****************************************************************
v2428com
v2429com****************************************************************
v2430com
v2431com single precision unbuffered i/o (ross)
v2432com
v2433com****************************************************************
v2434com
v2435com****************************************************************
v2436s.sensevent (signam :1,1,0,8: 0,40,11,3,8,2436,2445)
v2437com primitive to detect event-type input
v2438com list = source :max-input-lines,max-events: storage,time,ext,calc,incl,
v2439calc evpnt =evpnt + 1
v2440calc evaddr = evpnt *8
v2441begin stext
v2442; value assigned to <signam> by interrupt <evpnt>
v2443endtext
v2444incl h.sensevent ( <signam>, <evpnt> :1)
```

```
v2445com*****************************************************************
v2446s.eventmark (signam,evaddr ::3,15,3,8,0,2446,2455)
v2447com primitive to establish interrupt handler for event
v2448com list=source,trap-addr:empty:storage,time,ext,calc,incl,addr
v2449begin stext
v2450;
v2451        org    <evaddr>     ; interrupt trap for <signam>
v2452        mov    <signam>,1
v2453endtext
v2454calc romptr = romptr + 3
v2455com*****************************************************************
v2456h.sensevent (signam,evpnt : 1,8,0,128: 2,500,1,22,21,2456,2479)
v2457com primitive to define event-type input hardware
v2458com list= source, port-number: min/max-input-lines, min/max-events:
v2459com          latency,power,calc,includes,address
v2460 begin htext
v2461  event-mode input interface hardware to sense signal: <signam>
v2462   device: intel 8212 8-bit i/o port, ic <icn>
v2463   connections:
v2464   note: '1' = +5v, '0' = gnd
v2465    pins 3,5,6 (di(1:3)) = '111'
v2466    pins 20,22 (di(7:8)) = '11'
v2467    pins 9,16,18 (di(4:6)) = ' format(evpnt,b,3) '
v2468    pins 4,6,8,10,15,17,19,21 (do(1:8)) = db(1:8)
v2469    pin  2 (md) = '1'
v2470    pin 11 (stb) ='0'
v2471    pin  1 (ds1-bar) = .not. (inta .and. <signam>)
v2472    pin 13 (ds2) = dbin
v2473    or-tie <signam> to int
v2474    pin 12 (gnd) = gnd
v2475    pin 24 (vcc) =+5v
v2476 endtext
v2477 incl s.eventmark ( <signam>, <evaddr>:)
v2478calc icn=icn+1
v2479com*****************************************************************
v2480s.issuevent (signam:0,8,0,128: 5,23,4,0,4,2480,2485)
v2481com primitive to send event-type output
v2482com list=sink: output-lines,,events,,:storage,time,ext,calc,incl,addr
v2483com use same technique as conditional issue
v2484incl s.issuecond ( <signam>:8)
v2485com*****************************************************************
v2486h.issuevent (signam,outprt:0,8,0,128: 2,500,1,0,5,2486,2492)
v2487com primitive to define event-type output hardware
v2488com list= sink,port-num : min/max-output-lines, min/max-events
v2489com         latency, power,calc,incl,addr
v2490com   use same technique for both condition and event output
v2491 incl h.issuecond (signam,outprt:)
v2492com*****************************************************************
v2493h.issuecond (signam,:0,8,0,128: 2,500,1,17,0,2493,2519)
v2494com primitive to define condition-type output hardware
v2495com list= sink, : min/nax-output-lines,min/max-events:
v2496com         latency, power,calc,incl,addr
```

```
v2497 begin htext
v2498  condition-mode output interface hardware to issue signal: <signam>
v2499   device: intel 8212 8-bit i/o port, ic <icn>
v2500   connections:
v2501    pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
v2502    pins 4,6,8,10,15,17,19,21 (do(1:8)) = <signam>(1:8) ;if 8 are req
v2503    pin  2 (md) = +5v
v2504    pin 11 (stb) = gnd
v2505    pin  1 (ds1-bar) = wr-bar
v2506    pin 13 (ds2) = out .and. (decode a(0:7) value <outprt>)
v2507    pin 24 (vcc) = +5v
v2508    pin 12 (gnd) = gnd
v2509 endtext
v2510calc icn=icn+1
v2511com****************************************************************
v2512com
v2513com****************************************************************
v2514com
v2515com  double precision i/o realizations (pollock)
v2516com
v2517com****************************************************************
v2518com
v2519com****************************************************************
v2520h.issuecond (signam:9,16,0,32000:0,0,0,11,10,2520,2541)
v2521com primitive to define double precision condition-type
v2522com output hardware
v2523com list=signal:bits,conditions:lat,pwr,chips,calc,incl,addr
v2524name $na081-$na096
v2525begin htext
v2526  16 bit output port composed of two 8 bit ports
v2527      <$na081> is for low order byte
v2528      <$na089> is for high order byte
v2529endtext
v2530incl h.issuecond ( <$na081>, <outprt>:8,128)
v2531calc outprt=outprt+1
v2532incl h.issuecond ( <$na089>, <outprt>:8,128)
v2533com****************************************************************
v2534com
v2535com****************************************************************
v2536com
v2537com  digital i/o, optically isolated (pollock)
v2538com
v2539com****************************************************************
v2540com
v2541com****************************************************************
v2542s.sensopevt (signam,sigret:1,1,0,8:0,0,0,0,5,2542,2550)
v2543com primitive to define optically isolated event input
v2544com list=signal,return:lines,events:stor,time,ext,calc,incl,addrp
v2545name $na001
v2546name $na002
v2547incl h.resmfqtrwt( <$na002>, <$na001>,180:180:)
v2548incl h.optoisol  ( <$na001>, <sigret>, <signam>::)
```

```
v2549call s.sensevent ( <signam>:1,1:)
v2550com**************************************************************
v2551s.sensopcond(signam:0,8,0,128:0,0,0,0,4,2551,2573)
v2552com primitive to define optically isolated condition input
v2553com list=input name::stor,time,ext,calc,incl,addrs
v2554name $na001-$na036
v2555incl h.rpack-18b ( <$na001>, <$na017>::)
v2556incl h.rpack-18b ( <$na001>, <$na018>::)
v2557incl h.rpack-18b ( <$na003>, <$na019>::)
v2558incl h.rpack-18b ( <$na004>, <$na020>::)
v2559incl h.rpack-18b ( <$na005>, <$na021>::)
v2560incl h.rpack-18b ( <$na006>, <$na022>::)
v2561incl h.rpack-18b ( <$na007>, <$na023>::)
v2562incl h.rpack-18b ( <$na008>, <$na024>::)
v2563incl h.optoisol  ( <$na017>, <$na016>, <signam>(1)
v2564incl h.optoisol  ( <$na018>, <$na015>, <signam>(2)
v2565incl h.optoisol  ( <$na019>, <$na014>, <signam>(3)
v2566incl h.optoisol  ( <$na020>, <$na013>, <signam>(4)
v2567incl h.optoisol  ( <$na021>, <$na012>, <signam>(5)
v2568incl h.optoisol  ( <$na022>, <$na011>, <signam>(6)
v2569incl h.optoisol  ( <$na023>, <$na010>, <signam>(7)
v2570incl h.optoisol  ( <$na024>, <$na009>, <signam>(8)
v2571incl h.ncon16   ( <signam>
v2572call s.sensevent( <signam>:8,128:)
v2573com**************************************************************
v2574s.issueopcnd(signam:0,8,0,128:0,0,0,0,4,2574,2596)
v2575com primitive to define optically isolated digital output
v2576com suitable for operation of remote relays
v2577name $na001-$na024
v2578incl h.rpack-220 ( <signam>(1),+5v::)
v2579incl h.rpack-220 ( <signam>(2),+5v::)
v2580incl h.rpack-220 ( <signam>(3),+5v::)
v2581incl h.rpack-220 ( <signam>(4),+5v::)
v2582incl h.rpack-220 ( <signam>(5),+5v::)
v2583incl h.rpack-220 ( <signam>(6),+5v::)
v2584incl h.rpack-220 ( <signam>(7),+5v::)
v2585incl h.rpack-220 ( <signam>(8),+5v::)
v2586incl h.opisol2   ( <signam>(1),gnd, <$na001>, <$na016>::)
v2587incl h.opisol2   ( <signam>(2),gnd, <$na002>, <$na015>::)
v2588incl h.opisol2   ( <signam>(3),gnd, <$na003>, <$na014>::)
v2589incl h.opisol2   ( <signam>(4),gnd, <$na004>, <$na013>::)
v2590incl h.opisol2   ( <signam>(5),gnd, <$na005>, <$na012>::)
v2591incl h.opisol2   ( <signam>(6),gnd, <$na006>, <$na011>::)
v2592incl h.opisol2   ( <signam>(7),gnd, <$na007>, <$na010>::)
v2593incl h.opisol2   ( <signam>(8),gnd, <$na008>, <$na009>::)
v2594incl h.ncon16    ( <signam>:16:)
v2595call s.issuecond ( <signam>:8,128)
v2596com**************************************************************
v2597h.optoisol  (signam,sigret,sigout::0,95,1,19,4,2597,2617)
v2598com primitive to define optically isolated logic device
v2599com list=signal,return,output:empty:lat,pwr,chps,calc,incl,addrs
v2600com device is o.c. output, so a pull up resistor is included
```

```
v2601incl h.resmfqtrwt( <sigout>,+5v,3900:3900:)
v2602begin htext
v2603  optical isolator
v2604    device is hewlett packard hcpl-2602, ic <icn>
v2605      connections:
v2606        pin 1 = n.c.
v2607        pin 2 =  <signam0
v2608        pin 3 =  <sigret0
v2609        pin 4 = n.c. (led cathode, not used in this applic.)
v2610        pin 5 = gnd
v2611        pin 6 =  <sigout0
v2612        pin 7 = n.c. (enable, h.p. says it's ok to float)
v2613        pin 8 = +5v
v2614      ceramic bypass cap at chip is mandatorp
v2615endtext
v2616calc icn=icn+1
v2617com+*************************************************
v2618h.opisol2   (in,ret,outcol,outem::0,110,1,15,0,2618,2642)
v2619com primitive to define optical isolator,slow photo-darlington
v2620com list = input signal, return,output collector,emitter:;
v2621com        lat,pwr,chips,calc,incl,addrs
v2622begin htext
v2623  optical isolator with darlington output, ic  <icn>
v2624    device is 4n32
v2625    connections:
v2626      pin 1 =  <in>  (led anode)
v2627      pin 2 =  <ret> (led cathode)
v2628      pin 3 = n.c.
v2629      pin 4 =  <outem> (emitter)
v2630      pin 5 =  <outcol> (collector)
v2631      pin 6 = n.c.     (base)
v2632endtext
v2633calc icn = icn + 1
v2634com*****************************************************************
v2635com
v2636com*****************************************************************
v2637com
v2638com  individual single bit i/o realizations (pollock)
v2639com
v2640com*****************************************************************
v2641com
v2642com*****************************************************************
v2643s.relayout  (s,v:0,2:0,0,0,0,4,2643,2650)
v2644com relay output
v2645com list = sname,bufname:current:stor,time,ext,calc,incl,addrs
v2646name $na001-$na002
v2647incl h.relaydpdt ( <$na002>,+24vdc,<s>:2:)
v2648incl h.peripdrivr( <s>,true, <$na002>::)
v2649call s.issuesepcd( <s>, <v>:8,127:)
v2650com*****************************************************************
v2651s.issuesepcd(s,v:0,8,0,128:4,23,5,5,7,2651,2667)
v2652com issue separate single bit condition
```

```
v2653com list=sname,bufname:bits,states:stor,time,ext,calc,incl,addrs
v2654com every c,t independently writes the port
v2655if isce .ne. 1 skip 2
v2656calc iscp = outprt
v2657calc outprt = outprt +1
v2658call s.maskrepl  ( <v>, <isce>, <s>:8:)
v2659com variable v, declared elsewhere, is the one word
v2660com buffer associated with this output port
v2661begin stext
v2662         mov   AL,<v>         ;output logical  <s> from port  <iscp>
v2663         out  <iscp>      ;  bit  <isce>
v2664endtext
v2665calc romptr=romptr+5
v2666incl h.issuesepcd( <s>:8,128:)
v2667com********************************************************************
v2668h.issuesepcd(s:0,8,0,128:2,0,0,4,0,2668,2746)
v2669com single bit output
v2670com list=name:bits,states:lat,pwr,chips,calc,incl,addrs
v2671if isce .ne. 1 skip 4
v2672calc iscn = icn
v2673calc icn =icn+1
v2674attr pwr = pwr + 500
v2675attr chips = chips + 1
v2676begin htext
v2677  condition mode output, sliced bitwise
v2678     ic <iscn>, element  <isce>, for signal  <s>
v2679endtext
v2680if isce .eq. 1 skip 8
v2681if isce  .eq. 2 skip 21
v2682if isce  .eq. 3 skip 26
v2683if isce  .eq. 4 skip 31
v2684if isce  .eq. 5 skip 36
v2685if isce  .eq. 6 skip 41
v2686if isce .eq. 7 skip 46
v2687if isce .eq. 8 skip 51
v2688com element 1
v2689begin htext
v2690    device is intel 8212 8 bit i/o port
v2691     connections:
v2692      pin 24 = +5v     (vcc)
v2693      pin 12 = grd     (grd)
v2694      pin  2 = true    (md)
v2695      pin 11 = grd     (stb)
v2696      pin  1 = wr-bar  (dsl-bar)
v2697      pin 13 = out .and. (decode a(0:7) value  <iscp>
v2698      pin 3,5,7,9,16,18,20,22 = (di(1:8)) = db(1:8)
v2699      pin 4  =  <s>    ;(output)
v2700endtext
v2701skip 43
v2702com element 2
v2703begin htext
v2704       connections:
```

```
v2705         pin  6 =  <s>     (signal output)
v2706endtext
v2707skip 37
v2708com element 3
v2709begin htext
v2710       connection:
v2711          pin  8 =  <s>     (signal output)
v2712endtext
v2713skip 31
v2714com element 4
v2715begin htext
v2716       connection:
v2717         pin 10 =  <s>     (signal output)
v2718endtext
v2719skip 25
v2720com element 5
v2721begin htext
v2722     connection:
v2723       pin 15 =  <s>     (signal output)
v2724endtext
v2725skip 19
v2726com element 6
v2727begin htext
v2728     connection:
v2729       pin 17 =  <s>     (signal output)
v2730endtext
v2731skip 13
v2732com element 7
v2733begin htext
v2734     connection:
v2735       pin 19 =  <s>     (signal output)
v2736endtext
v2737skip 7
v2738com element 8
v2739begin htext
v2740     connection:
v2741       pin 21 =  <s>     (signal output)
v2742endtext
v2743calc isce = 0
v2744com common end
v2745calc isce =isce+1
v2746com*******************************************************************
v2747h.peripdrivr(a,b,o::0,0,0,4,0,2747,2779)
v2748com peripheral driver
v2749com list=inputs a,b,output::lat,pwr,chips,calc,incl,addrs
v2750if pde .ne. 1 skip 18
v2751calc pdn = icn
v2752calc icn = icn + 1
v2753attr pwr = pwr + 325
v2754attr chips = chips + 1
v2755com element 1
v2756begin htext
```

```
v2757  peripheral driver for signal  <o>
v2758  ic <pdn>, element 1
v2759  device is sn75452
v2760    connections:
v2761      pin  8 = +5v     (vcc)
v2762      pin  4 = pdgrd   (peripheral ground)
v2763      pin  1 =  <a>     (input 1)
v2764      pin  2 =  <b>     (input 2)
v2765      pin  3 =  <o>    ;(output)
v2766endtext
v2767skip 11
v2768com element 2
v2769begin htext
v2770  element 2 of peripheral driver ic <pdn>, for signal  <o>
v2771    connections:
v2772      pin  6 =  <a>     (input 1)
v2773      pin  7 =  <b>     (input 2)
v2774      pin  5 =  <o>    ;(output)
v2775endtext
v2776calc pde = 0
v2777com common ending
v2778calc pde = pde + 1
v2779com*******************************************************************
v2780h.relaydpdt (s,v,t:0,2:0,2000,0,16,15,2780,2797)
v2781com relay, 2 amp form c contents, 1/2 size xtal can
v2782com list = signal,title:amps:lat,pwr,chips,calc,incl,addrs
v2783begin htext
v2784  relay k <kn>, 1/2 size xtal can, 24v coil, 2a contacts form c
v2785     connections:
v2786       pin   3=  <s>      (coil neg)
v2787       pin   7=  <v>      (coil pos)  (blue dot)
v2788       pin   6=  <t>-cl   (contacts)
v2789       pin   8=  <t>-ncl
v2790       pin   1=  <t>-nol
v2791       pin   2=  <t>-c2
v2792       pin   4=  <t>-nc2
v2793       pin   5=  <t>-no2
v2794endtext
v2795incl h.diode     ( <v>, <s>::)
v2796calc kn=kn+1
v2797com*******************************************************************
v2798s.senscontac(s,b::0,0,0,0,6,2798,2810)
v2799com sense contact closure
v2800begin stext
v2801;  sense contact closure  <s>
v2802endtext
v2803name $na001-$na002
v2804incl h.conn26sep ( <$na002>,grd,chass,<s>::)
v2805incl h.resmfqtrwt( <$na002>,+5v,1000:1000)
v2806incl h.resmfqtrwt( <$na002>, <$na001>,1000:1000:)
v2807incl h.capac-cer ( <$na001>,grd,6200000:6200000)
v2808incl h.striginvrt( <$na001>, <s>:)
```

```
v2809call s.sensepcond( <s>,<b>:8,128:)
v2810com*******************************************************************
v2811s.senshotct (s1,s2,v,m,b:10,60:0,0,0,9,8,2811,2827)
v2812com sense hot contact
v2813com list= signal,volts across open contacts, max ma,buf:ma lims:
v2814com       stor,time,ext,calc,incl,addrs
v2815begin stext
v2816sense hot contact closure  <s1>, <s2>
v2817endtext
v2818name $na001-$na004
v2819incl h.conn26sep ( <$na004>, <s2>,chas,<s1>::)
v2820calc scrtch = 100 * v
v2821incl h.resmfqtrwt( <$na004>, <$na001>, <scrtch>: <scrtch>:)
v2822incl h.optoisol  ( <$na001>, <s2>, <$na002>::)
v2823incl h.resmfqtrwt( <$na002>, <$na003>,1000:1000:)
v2824incl h.striginvrt( <$na003>, <s1>::)
v2825incl h.capac-cer ( <$na003>,grd,6200000:6200000:)
v2826call s.sensepcond( <s1>,<b>:8,128)
v2827com*******************************************************************
v2828s.sensepcond(s,b:0,8,0,128:5,25,4,5,11,2828,2842)
v2829com sense separate single bit condition
v2830com list=name,buffer:bits,states:stor,time,ext,calc,incl,addrs
v2831com every c,t independently reads the port
v2832if ssche .ne. 1 skip 2
v2833calc inport = inport+1
v2834calc sscp = inport
v2835begin stext
v2836        in    <sscp>       ;read buffer word  <b> from port  <sscp>
v2837        mov   <b>,AL
v2838endtext
v2839call s.mask      ( <s>,<b>, <ssche>:8,8,8:)
v2840incl h.sensepcond( <s>:8,128:)
v2841calc romptr = romptr + 5
v2842com*******************************************************************
v2843h.sensepcond(s:0,8,0,128:2,0,0,4,0,2843,2919)
v2844com single bit input
v2845com list=name:bits,states:lat,pwr,chips,calc,incl,addrs
v2846if ssche .ne. 1 skip 2
v2847calc sschn = icn
v2848calc icn = icn+1
v2849begin htext
v2850  condition-mode input, sliced bitwise
v2851    ic <sschn>, element  <ssche>, for signal  <s>
v2852endtext
v2853if ssche .eq. 1 skip 8
v2854if ssche .eq. 2 skip 21
v2855if ssche .eq. 3 skip 26
v2856if ssche .eq. 4 skip 31
v2857if ssche .eq. 5 skip 36
v2858if ssche .eq. 6 skip 41
v2859if ssche .eq. 7 skip 46
v2860if ssche .eq. 8 skip 51
```

```
v2861com element 1
v2862begin htext
v2863    device is intel 8212 8 bit i/o port
v2864     connections:
v2865      pin 24 = +5v      (vcc)
v2866      pin 12 = grd      (grd)
v2867      pin  2 = grd      (md)
v2868      pin 11 = grd      (stb)
v2869      pin  1 = .not. (decode a(0:7) value  <sscp>      (ds1bar)
v2870      pin 13 = inp .and. dbin    (ds2)
v2871      pins 4,6,8,10,15,17,19,21 (do(1:8)= db(1:8)
v2872      pin  3 =  <s>     (signal input,di(1))
v2873endtext
v2874skip 43
v2875com element 2
v2876begin htext
v2877     connection:
v2878       pin  5 =  <s>      (signal input, di(2))
v2879endtext
v2880skip 37
v2881com element 3
v2882begin htext
v2883     connection:
v2884       pin  7 =  <s>      (signal input, di(3))
v2885endtext
v2886skip 31
v2887com element 4
v2888begin htext
v2889     connection:
v2890       pin  9 =  <s>      (signal input, di(4))
v2891endtext
v2892skip 25
v2893com element 5
v2894begin htext
v2895     connection:
v2896       pin 16 =  <s>      (signal input, di(5))
v2897endtext
v2898skip19
v2899com element 6
v2900begin htext
v2901     connection:
v2902       pin 18 =  <s>      (signal input, di(6))
v2903endtext
v2904skip 13
v2905com element 7
v2906begin htext
v2907     connection:
v2908       pin 20 =  <s>      (signal input, di(7))
v2909endtext
v2910com element 8
v2911begin htext
v2912     connection:
```

```
v2913       pin 22 =  <s>      (signal input, di(8))
v2914endtext
v2915skip 2
v2916calc ssche = 0
v2917com common end
v2918calc ssche =ssche + 1
v2919com*******************************************************************
v2920h.conn26sep (s,r,sh,t::0,0,0,4,0,2920,3005)
v2921com 26 pin flat cable connector expressed as 8 elements of 3 pinp
v2922com list = signal,rtn,shld,title::lat,pwr,chips,calc,incl,addrs
v2923if jaase  .ne. 1 skip 11
v2924calc jaasn = jn
v2925calc jn = jn+1
v2926begin htext
v2927  connector j <jaasn>, element  <jaase >
v2928   device is 26 pin flat cable connector
v2929    connections:
v2930      pin  1 =  <s>
v2931      pin 14 =  <r>
v2932      pin  2 =  <sh>
v2933endtext
v2934skip 61
v2935begin htext
v2936  part of connector j <jaasn>, for signal <t0
v2937    connections:
v2938endtext
v2939if jaase  .eq. 2  skip 7
v2940if jaase  .eq. 3  skip 10
v2941if jaase  .eq. 4  skip 10
v2942if jaase  .eq. 5  skip 20
v2943if jaase  .eq. 6  skip 30
v2944if jaase  .eq. 7  skip 30
v2945if jaase  .eq. 8 skip 43
v2946com element 2
v2947begin htext
v2948      pin 16 =  <s>
v2949      pin 15 =  <r>
v2950      pin  3 =  <sh>
v2951endtext
v2952skip 43
v2953com element 3
v2954begin htext
v2955      pin 18 =  <s>
v2956      pin 17 =  <r>
v2957      pin  4 =  <sh>
v2958endtext
v2959skip 36
v2960com element 4
v2961begin htext
v2962      pin  6 =  <s>
v2963      pin  5 =  <r>
v2964      pin 19 =  <sh>
```

```
v2965endtext
v2966skip 29
v2967com element 5
v2968begin htext
v2969      pin  7 =  <s>
v2970      pin 20 =  <r>
v2971      pin  8 =  <sh>
v2972endtext
v2973skip 22
v2974com element 6
v2975begin htext
v2976      pin 22 =  <s>
v2977      pin 21 =  <r>
v2978      pin  9 =  <sh>
v2979endtext
v2980skip 15
v2981com element 7
v2982begin htext
v2983      pin 24 =  <s>
v2984      pin 23 =  <r>
v2985      pin 10 =  <sh>
v2986endtext
v2987skip 8
v2988com element 8
v2989begin htext
v2990      pin 12 =  <s>
v2991      pin 11 =  <r>
v2992      pin 25 =  <sh>
v2993endtext
v2994calc jaase  = 0
v2995com common ending
v2996calc jaase =jaase +1
v2997com*******************************************************************
v2998com
v2999com*******************************************************************
v3000com
v3001com  analog i/o realizations (pollock)
v3002com
v3003com*******************************************************************
v3004com
v3005com*******************************************************************
v3006s.analogin  (sig,h,l,b:0,8,100,-100,1,100:0,0,0,0,10,3006,3050)
v3007com primitive to define analog input condition
v3008com list=(input signal,high voltage limit, low voltage limit,
v3009com      3db rolloff:bits,volt limits,bw limits:
v3010com      time,stor,ext,calc,incl,addrs
v3011name $na001-$na050
v3012begin htext
v3013analog input channel for signal  <sig>, range  <h> to  <l>,
v30143db rolloff at  <b>
v3015endtext
v3016incl h.conn-al   (<$na006>,<$na007>,grd,<sig>::)
```

```
v3017com select gain for buffer amp to match range
v3018if  <h> .ge. 50 skip 9
v3019if  <h> .eq. 20 skip 2
v3020if  <l> .eq. 0  skip 4
v3021com  gain 2.5 (expressed 25) for +-2;+-1;0,+2 ranges
v3022incl h.bufframp  ( <$na006), <$na007>, <$na005>,25, <b>::)
v3023skip 6
v3024com  gain 5.0 (expressed 50) for 0,+1 range
v3025incl h.bufframp  ( <$na006), <$na007>, <$na005>,50, <b>::)
v3026skip 3
v3027com  gain 1.0 (expressed 10) for 0,+5;0,+10,+-5,+-10 ranges
v3028incl h.bufframp  ( <$na006), <$na007>, <$na005>,10, <b>::)
v3029com select adc to match range
v3030if  <l> .eq. -10 skip 16
v3031if  <l> .le. -2  skip 12
v3032if  <l> .eq. -1  skip 8
v3033if  <l> .eq. 10  skip 4
v3034 com adc range 0,+5 for 0,+1;0,+2;0,+5 input ranges
v3035 incl h.adc       ( <$na005>, <sig>,50,0:8:)
v3036 skip 11
v3037 com adc range 0,+10 for 0,+10 input range
v3038 incl h.adc       ( <$na005>, <sig>,100,0:8:
v3039 skip 8
v3040 com adc range +-2.5 for +-2.5 input range
v3041 incl h.adc       ( <$na005>, <sig>,25,-25:8:)
v3042 skip 5
v3043 com adc range +-5 for +-2,+-5 input ranges
v3044 incl h.adc       ( <$na005>, <sig>,50,-50:8:)
v3045 skip 2
v3046 com adc range +-10 for +-10 input range
v3047 incl h.adc       ( <$na005>, <sig>,100,-100:8:)
v3048 com
v3049 call s.sensecond ( <sig>:8,128)
v3050com*******************************************************************
v3051s.anaout    (sigout,h,l:0,8,25,100,0,-100:0,0,0,0,10,3051,3073)
v3052com primitive to define analog output channel
v3053com list =signal out,bits,hi and lo voltage limits:
v3054com      ranges for b,h,l:
v3055com      time,stor,ext,calc,incl,addrs
v3056com note voltages in 100 mv units
v3057begin htext
v3058analog output channel for signal  <sigout>, range  <h> to  <l>
v3059endtext
v3060name$na051-$na070
v3061incl h.conn-al   ( <$na060>,gnd,open,<sigout>::)
v3062incl h.follower  ( <$na051>, <sigout>::)
v3063incl h.dac       ( <$na052>, <$na051>,50,-50:8:)
v3064incl h.invert    ( <sigout>(1), <$na052>::)
v3065incl h.invert    ( <$na061>, <$na053>::)
v3066incl h.invert    ( <$na062>, <$na054>::)
v3067incl h.invert    ( <$na063>, <$na055>::)
v3068incl h.invert    ( <$na064>, <$na056>::)
```

```
v3069incl h.invert    ( <$na065>, <$na057>::)
v3070incl h.invert    ( <$na066>, <$na058>::)
v3071incl h.invert    ( <$na067>, <$na059>::)
v3072call s.issuecond ( <$na060>:8,128:)
v3073com******************************************************************
v3074h.adc       (in,out,h,l:0,8,25,100,0,-100:0,400,1,93,89,3074,3168)
v3075com primitive to define 8 bit adc
v3076com list= analog input,digital output,high v limit, lo v limit:
v3077com       bits:lat,pwr,chips,calc,incl,addrs
v3078begin htext
v3079  a/d convertor, 8 bit
v3080    device is burr brown adc82ag, ic <icn>
v3081      connections:
v3082        pin  1 = n.c.    (clock out)
v3083        pin  2 = gnd     (digital common)
v3084        pin  3 = n.c.    (status)
v3085        pin  4 =  <out>(0) (lsb)
v3086        pin  5 =  <out>(1) (2lsb)
v3087        pin  6 =  <out>(2) (3lsb)
v3088        pin  7 =  <out>(3) (4lsb)
v3089        pin  8 =  <out>(4) (5lsb)
v3090        pin  9 =  <out>(5) (6lsb)
v3091        pin 10 =  <out>(6) (7lsb)
v3092endtext
v3093com test for unipolar connection. change code if so
v3094if   <l> .ne. 0 skip 5
v3095begin htext
v3096        pin 11 =  <out>(7) (msb)  (comp bin code for unipolar)
v3097        pin 12 = n.c.             (not msb, unused for this code)
v3098endtext
v3099skip 4
v3100begin htext
v3101       pin 12 =  <out> (7) msb not
v3102       pin 11 = n.c.
v3103endtext
v3104begin htext
v3105        pin 13 =  <$na021>         (gain adjust)
v3106endtext
v3107com set input and jumpers for voltage scale (voltages in 100mv unit
v3108if  <l> .eq. -100 skip 5
v3109if  <l> .eq. -50  skip 12
v3110if  <l> .eq. -25  skip 19
v3111if  <h> .eq.  50  skip 25
v3112if  <h> .eq.  100 skip 32
v3113com +-10v adc range
v3114begin htext
v3115        pin 14 = n.c.            (10 v range input)
v3116        pin 15 =  <in>            (20 v range input)
v3117        pin 16 jumper to pin 18
v3118        pin 17 = analog ground
v3119endtext
v3120skip 31
```

```
v3121com +-5v adc range
v3122begin htext
v3123         pin 14 =  <in>       (10 v range input)
v3124         pin 15 = n.c.       (20 v range input
v3125         pin 16 jumper to pin 18
v3126         pin 17 = analog ground
v3127endtext
v3128skip 23
v3129com +-2.5v adc range
v3130begin htext
v3131         pin 14 =  <in>       (10 v range input)
v3132         pin 15 = (jmp)   ;jumper to pins 16 and 18
v3133         pin 17 = (agnd)  ;analog ground
v3134endtext
v3135skip 16
v3136com 0,+5v adc range
v3137begin htext
v3138         pin 14 =  <in>            (10 v range input)
v3139         pin 15 jumper to pin 18
v3140         pin 16 = analog ground
v3141         pin 17 = analog ground
v3142endtext
v3143skip 8
v3144com 0,+10v adc range
v3145begin htext
v3146         pin 14 =  <in>       (10 v range input)
v3147         pin 15 = n.c.       (20 v range input
v3148         pin 16 = analog ground
v3149         pin 17 = analog ground
v3150endtext
v3151com
v3152begin htext
v3153         pin 18 = n.c.
v3154         pin 19 = +15v
v3155         pin 20 = -15v
v3156         pin 21 = n.c.          (serial data)
v3157         pin 22 = ph2           (clock)
v3158         pin 23 = n.c.          (convert)
v3159         pin 24 = +5v
v3160       connect digital ground (pin 2) and analog ground (pin 17)
v3161       at one place only, at this chip
v3162endtext
v3163incl h.resmfqtrwt(<$na021>,<$na022>,1000000:1000000)
v3164incl h.trimpot   (+15v,-15v, <$na022>,100000:100000:)
v3165incl h.resmfqtrwt( <$na023>, <$na024>,1000000:1000000:)
v3166incl h.trimpot   (+15v,-15v, <$na024>,100000:100000:)
v3167calc icn=icn+1
v3168com*******************************************************************
v3169h.dac       (in,out,h,1:0,8:0,350,1,68,0,3169,3238)
v3170com primitive to define 8 bit
v3171com list = input, output,hi volt limit, lo volt limit:
v3172com        range of b,h,l:lat,pwr,chips,calc,incl,addrs
```

```
v3173com note voltages are in units of 100mv
v3174begin htext
v3175  8 bit dac, ic <icn>
v3176   device is burr-brown dac82. laser trimmed, no adj rqd.
v3177     connections:
v3178       pin 3  = +15v
v3179       pin 4  =  <in>(7)
v3180       pin 5  =  <in>(6)
v3181       pin 6  =  <in>(5)
v3182       pin 7  =  <in>(4)
v3183       pin 8  =  <in>(3)
v3184       pin 9  =  <in>(2)
v3185       pin 10 =  <in>(1)
v3186       pin 11 =  <in>(0)
v3187       pin 12 jumper to pin 15 to use internal current ref
v3188       pin 13 = +15v
v3189       pin 14 = gnd
v3190     range dependent connections for  <h>00 to  <l>00 mv range
v3191endtext
v3192com range dependent connections
v3193if <l> .eq. -100 skip 5
v3194if <l> .eq. -50  skip 12
v3195if <l> .eq. -25  skip 18
v3196if <h> .eq. 50   skip 25
v3197if <h> .eq. 100  skip 32
v3198com +/- 10 v range
v3199begin htext
v3200       pin 1 jumper to pin 18
v3201       pin 2 =  <out>      ;(output)
v3202       pin 16 jumper to pin 2
v3203       pin 17 = n.c.
v3204endtext
v3205skip 31
v3206com +/- 5v range
v3207begin htext
v3208       pin 1 jumper to pin 18
v3209       pin 2 =  <out>      ;(output)
v3210       pin 16= n.c.
v3211       pin 17 jumper to pin 2
v3212endtext
v3213skip 23
v3214com +/- 2.5 v range
v3215begin htext
v3216       pin 1  jumper to pins 16 and 18
v3217       pin 2 =  <out>         ;(output)
v3218       pin 17 jumper to pin 2
v3219endtext
v3220skip 16
v3221com 0,+5 v range
v3222begin htext
v3223       pin 1  = gnd
v3224       pin 2  =  <out>        ;(output)
```

```
v3225       pin 16 jumper to pin 18
v3226       pin 17 jumper to pin 2
v3227endtext
v3228skip 8
v3229com 0,+10 v range
v3230begin htext
v3231       pin 1  = gnd
v3232       pin 2  =  <out>        ;(output)
v3233       pin 16 = n.c.
v3234       pin 17 jumper to pin 2
v3235       pin 18 = n.c.
v3236endtext
v3237calc icn=icn+1
v3238com******************************************************************
v3239h.follower  (in,out::0,150,1,16,0,3239,3256)
v3240com primitive to define a voltage follower
v3241com list = input, output::lat,pwr,chips,calc,incl,addrs
v3242begin htext
v3243  voltage follower, ic<icn>
v3244    device is lm310
v3245     connections:
v3246       pin 1 = n.c.       (balance)
v3247           2 = n.c.
v3248           3 =  <in>      ;(input)
v3249           4 = -15v       (v-)
v3250           5 = n.c.       (booster)
v3251           6 =  <out>     ;(output)
v3252           7 = +15v       (v+)
v3253           8 = n.c.       (balance. tab at pin 8)
v3254endtext
v3255calc icn=icn+1
v3256com******************************************************************
v3257h.opamp     (p,n,o,t,u::0,85,1,17,0,3257,3275)
v3258com primitive to define operational amplifiep
v3259com list=positive signal in,negative,output,zero pot end 1,end 2::
v3260com      lat,pwr,chp,calc,incl,addrs
v3261begin htext
v3262  op amp, ic  <icn>
v3263    device is analog devices ad741k
v3264     connections:
v3265       pin 1 =  <t>     (zero trimpot)
v3266           2 =  <n>     (negative input)
v3267           3 =  <p>     (positive input)
v3268           4 = -15v
v3269           5 =  <u>     (zero trimpot)
v3270           6 =  <o>    ;(output)
v3271           7 = +15v
v3272           8 = n.c.    tab at pin 8
v3273endtext
v3274calc icn = icn+1
v3275com******************************************************************
v3276h.bufframp  (in,ret,out,g,b::0,150,1,22,5,3276,3313)
```

```
v3277com primitive to define buffer input amplifier, 10k ohm input z
v3278com list=input,return,output,gain (v/v),bandwidth in khz at 3db:
v3279com      gain limits, bw limits
v3280com gain is stated a10. that is gain 2.5 is stated g=25
v3281incl h.resmfqtrwt( <in>, <$na001>,10000:10000:)
v3282incl h.resmfqtrwt( <ret>, <$na002>,10000:10000:)
v3283if  <g> .eq. 10 skip 3
v3284if  <g> .eq. 25 skip 6
v3285if  <g> .eq. 50 skip 9
v3286com gain = 1
v3287incl h.resmfqtrwt( <$na001>, <out>,10000:10000:)
v3288incl h.resmfqtrwt( <$na002>,gnd,10000:10000:
v3289skip 8
v3290com gain = 2.5
v3291incl h.resmfqtrwt( <$na001>, <out>,25000:25000:)
v3292incl h.resmfqtrwt( <$na002>,gnd,25000:25000:
v3293skip 4
v3294com gain = 5
v3295incl h.resmfqtrwt( <$na001>, <out>,50000:50000:)
v3296incl h.resmfqtrwt( <$na002>,gnd,50000:50000:
v3297com calculate bandwidth limiting capacitor
v3298calc scrtch = <b>*<g>
v3299calc scrtch = 160000/scrtch
v3300incl h.capac-cer ( <$na001>, <out>, <scrtch>: <scrtch>:)
v3301com add a zero pot
v3302incl h.trimpot   ( <$na003>, <$na004>,-15v,10000:10000)
v3303incl h.opamp     (<$na002>,<$na001>,<out>,<$na003>,<$na004>::)
v3304com end of realization
v3305com******************************************************************
v3306com
v3307com******************************************************************
v3308com
v3309com  transducer and special purpose i/o  (pollock)
v3310com
v3311com******************************************************************
v3312com
v3313com******************************************************************
v3314s.temp      (signam,hit,lot:-55,85,-55,85:0,0,0,0,12,3314,3335)
v3315com primitive to define temperature measurement channel
v3316com -55 deg c = 0.13v = 010b on 2 volt adc range
v3317com +85 deg c = 1.54v = 143b
v3318com 1 deg c = 10 mv
v3319com lsb = 15.6mv = 1.56 deg c
v3320com list=signam, hi temp limit, lo temp limit:restrictions on hi,lo
v3321com      limits:stor,time,ext,calc,incl,addrp
v3322name $na001-$na050
v3323begin htext
v3324  analog input channel to measure temperature <signam>
v3325endtext
v3326incl h.conn-al   (<$na009>,<$na008>,grd,<signam>::)
v3327incl h.follower  (<$na008>,<$na006>::)
v3328incl h.bufframp  (<$na006>,grd,<$na005>,50,1::)
```

```
v3329incl h.adc       (<$na005>,<signam>,50,-50:8:)
v3330call s.sensecond (<signam>:8,128)
v3331incl h.resmfqtrwt(<$na009>,<$na008>,5825:5825)
v3332incl h.resmfqtrwt(<$na008>,<$na010>,3525:3525)
v3333incl h.resmfqtrwt(<$na010>,+5v,3000:3000)
v3334incl h.diode-znr (<$na010>,grd,2:2)
v3335com*******************************************************************
v3336s.restrans  (name,res:1000,10000:0,0,0,0,6,3336,3354)
v3337com primitive to define resistance transducep
v3338com +/- 5v scals is used, so 0-177b corresponds to zero to full
v3339com range
v3340com list=name,resistance:res limits:stor,time,ext,calc,incl,addrp
v3341name $na001-$na050
v3342incl h.conn-al   (+5v,<$na006>,grd,<name>:)
v3343incl h.bufframp  (<$na006>,grd,<$na005>,10,1::)
v3344incl h.adc       (<$na005>,<signam>,50,-50:8:)
v3345call s.sensecond (<name>:8,128:)
v3346com*******************************************************************
v3347com
v3348com*******************************************************************
v3349com
v3350com    clocks and timers   (pollock)
v3351com
v3352com*******************************************************************
v3353com
v3354com*******************************************************************
v3355s.clock     (name,freq:16,16:0,0,0,0,4,3355,3364)
v3356com primitive to define a free running time clock
v3357com list=name,freq(*100 khz):bits:stor,time,ext,calc,incl,addrs
v3358name $na001-$na010
v3359incl h.oscxtal   (<$na006>,<freq>:<freq>)
v3360incl h.oneshot1  (n.c.,<$na007>,n.c.,<$na006>,20:20)
v3361call s.sensecond (<name>:16)
v3362com the above call names $na081-$na096
v3363incl h.clckctrs  (::)
v3364com*******************************************************************
v3365h.oscxtal   (name,freq:1,200:0,300,1,11,0,3365,3377)
v3366com modular crystal oscillator
v3367com list=name,freq(*100 khz):freq limits:lat,pwr,chips,calc,incl,addrs
v3368begin htext
v3369  crystal oscillator module, ic<icn>, for signal <name>
v3370    device is motorola
v3371      connections:
v3372        pin  1 = +5v  (vcc)
v3373        pin  2 = grd  (grd)
v3374        pin  3= <name>  ;(output)
v3375endtext
v3376calc icn=icn+1
v3377com*******************************************************************
v3378h.oneshot1  (a,b,c,d,name,time:20,20:0,125,1,0,23,3378,3403)
v3379com primitive to define 9601 ttl one shot, retriggerable, edge
v3380com triggered
```

```
v3381com list=inverting "ored" inputs a,b, non-inverting "anded" inputs c,d,
v3382com       signal name, duration (nsec):duration limits:
v3383com       lat,pwr,chips,calc,incl,addrs
v3384name $na001-$na002
v3385begin htext
v3386  ttl one shot, ic<icn>, for signal <name>, duration <time> nsec
v3387    device is 9601
v3388      connections:
v3389        pin  1 = <a>   (inverted "ored" inputs)
v3390        pin  2 = <b>
v3391        pin  3 = <c>   (non-inverted "anded" inputs)
v3392        pin  4 = <d>
v3393        pin  8 = <name>  ;(output)
v3394        pin  6 = .not. <name>   (.not. output)
v3395        pin 14 = +5v    (vcc)
v3396        pin  7 = grd    (grd)
v3397        pin 11 = <$na001>    (cx)
v3398        pin 13 = <$na002>    (rx/cx)
v3399endtext
v3400com note that only a 20 nsec version is avail now.
v3401incl hresmfqtrwt(<$na002>,;5v,5000:5000)
v3402incl h.capac-cer(<$na001>,<$na002>,20:20)
v3403com********************************************************************
v3404h.clckctrs  (::0,1000,1,24,0,3404,3503)
v3405com primitive to define 16 bit ttl counter for free running clock
v3406com list=::lat,pwr,chips,,,,,
v3407begin htext
v3408  4 bit binary counter, ic<icn>
v3409    device is sn74161n
v3410      connections:
v3411        pin  1 = +5v       (.not. clear)
v3412        pin  2 = <$na002>  (clock, rising edge active)
v3413        pin  3 = n.c.      (a input)
v3414        pin  4 = n.c.      (b input)
v3415        pin  5 = n.c.      (c input)
v3416        pin  6 = n.c.      (d input)
v3417        pin  7 = +5v       (enable p)
v3418        pin  8 = grd       (grd)
v3419        pin  9 = +5v       (.not. load)
v3420        pin 10 = +5v       (enable t)
v3421        pin 11 = <$na084>  (msb output)
v3422        pin 12 = <$na083>  (3lsb output)
v3423        pin 13 = <$na082>  (2lsb output)
v3424        pin 14 = <$na081>  (lsb output)
v3425        pin 15 = <$na003>  (carry output)
v3426        pin 16 = +5v       (vcc)
v3427endtext
v3428calc icn=icn+1
v3429begin htext
v3430  4 bit binary counter, ic<icn>
v3431    device is sn74161n
v3432      connections:
```

```
v3433        pin  1 = +5v       (.not. clear)
v3434        pin  2 = <$na002>  (clock, rising edge active)
v3435        pin  3 = n.c.      (a input)
v3436        pin  4 = n.c.      (b input)
v3437        pin  5 = n.c.      (c input)
v3438        pin  6 = n.c.      (d input)
v3439        pin  7 = <$na003>  (enable p)
v3440        pin  8 = grd       (grd)
v3441        pin  9 = +5v       (.not. load)
v3442        pin 10 = +5v       (enable t)
v3443        pin 11 = <$na088>  (msb output)
v3444        pin 12 = <$na087>  (3lsb output)
v3445        pin 13 = <$na086>  (2lsb output)
v3446        pin 14 = <$na085>  (lsb output)
v3447        pin 15 = <$na004>  (carry output)
v3448        pin 16 = +5v       (vcc)
v3449endtext
v3450calc icn=icn+1
v3451begin htext
v34524 bit binary counter, ic<icn>
v3453    device is sn74161n
v3454      connections:
v3455        pin  1 = +5v       (.not. clear)
v3456        pin  2 = <$na002>  (clock, rising edge active)
v3457        pin  3 = n.c.      (a input)
v3458        pin  4 = n.c.      (b input)
v3459        pin  5 = n.c.      (c input)
v3460        pin  6 = n.c.      (d input)
v3461        pin  7 = <$na004>    (enable p)
v3462        pin  8 = grd       (grd)
v3463        pin  9 = +5v       (.not. load)
v3464        pin 10 = +5v       (enable t)
v3465        pin 11 = <$na092>  (msb output)
v3466        pin 12 = <$na091>  (3lsb output)
v3467        pin 13 = <$na090>  (2lsb output)
v3468        pin 14 = <$na089>  (lsb output)
v3469        pin 15 = <$na005>  (carry output)
v3470        pin 16 = +5v       (vcc)
v3471endtext
v3472calc icn=icn+1
v3473begin htext
v3474  4 bit binary counter, ic<icn>
v3475    device is sn74161n
v3476      connections:
v3477        pin  1 = +5v       (.not. clear)
v3478        pin  2 = <$na002>  (clock, rising edge active)
v3479        pin  3 = n.c.      (a input)
v3480        pin  4 = n.c.      (b input)
v3481        pin  5 = n.c.      (c input)
v3482        pin  6 = n.c.      (d input)
v3483        pin  7 = <$na005>    (enable p)
v3484        pin  8 = grd       (grd)
```

```
v3485        pin  9 = +5v        (.not. load)
v3486        pin 10 = +5v        (enable t)
v3487        pin 11 = <$na096>   (msb output)
v3488        pin 12 = <$na095>   (3lsb output)
v3489        pin 13 = <$na094>   (2lsb output)
v3490        pin 14 = <$na093>   (lsb output)
v3491        pin 15 = n.c.     (carry output)
v3492        pin 16 = +5v        (vcc)
v3493endtext
v3494calc icn=icn+1
v3495com*******************************************************************
v3496com
v3497com*******************************************************************
v3498com
v3499com  logic elements (pollock)
v3500com
v3501com*******************************************************************
v3502com
v3503com*******************************************************************
v3504h.invert    (in,out::0,0,0,4,0,3504,3562)
v3505com primitive to define ttl invertor
v3506com list=input signal,output signal::lat,pwr,chips,calc,incl,addrs)
v3507if ine .ne. 1 skip 4
v3508calc inn=icn
v3509attr pwr = pwr + 60
v3510attr chips = chips + 1
v3511calc icn=icn+1
v3512begin htext
v3513  ttl invertor, element  <ine> of ic <inn>, 7404
v3514endtext
v3515if  <ine> .eq.1 skip 6
v3516if  <ine> .eq.2 skip 13
v3517if  <ine> .eq.3 skip 18
v3518if  <ine> .eq.4 skip 23
v3519if  <ine> .eq.5 skip 28
v3520if  <ine> .eq.6 skip 33
v3521com element 1
v3522begin
v3523    pin 1 =  <in>   ;(input)
v3524    pin 2 =  <out>  ;(output)
v3525    pin 7 = gnd
v3526    pin 14 = +5v
v3527endtext
v3528skip 32
v3529com element 2
v3530begin
v3531    pin 3 =  <in>    ;(input)
v3532    pin 4 =  <out>   ;(output)
v3533endtext
v3534skip 26
v3535com element 3
v3536begin
```

```
v3537    pin 5 =  <in>    ;(input)
v3538    pin 6 =  <out>   ;(output)
v3539endtext
v3540skip 20
v3541com element 4
v3542begin
v3543    pin 9 =  <in>    ;(input)
v3544    pin 8 =  <out>   ;(output)
v3545endtext
v3546skip 14
v3547com element 5
v3548begin
v3549    pin 11 =  <in>   ;(input)
v3550    pin 10 =  <out>  ;(output)
v3551endtext
v3552skip 8
v3553com element6
v3554begin
v3555    pin 13 =  <in>   ;(input)
v3556    pin 12 =  <out> ;(output)
v3557endtext
v3558com
v3559calc ine=0
v3560com common completion
v3561calc ine=ine+1
v3562com*******************************************************************
v3563h.striginvrt(in,out::0,0,0,4,0,3563,3621)
v3564com primitive to define ttl schmidt trigger invertor
v3565com list=input signal,output signal::lat,pwr,chips,calc,incl,addrs)
v3566if isne .ne. 1 skip 4
v3567calc isnn=icn
v3568attr pwr = pwr + 60
v3569attr chips = chips + 1
v3570calc icn=icn+1
v3571begin htext
v3572  ttl schmidt trigger invertor, element  <isne> of ic <isnn>, 7414
v3573endtext
v3574if  <isne> .eq.1 skip 6
v3575if  <isne> .eq.2 skip 13
v3576if  <isne> .eq.3 skip 18
v3577if  <isne> .eq.4 skip 23
v3578if  <isne> .eq.5 skip 28
v3579if  <isne> .eq.6 skip 33
v3580com element 1
v3581begin
v3582    pin 1 =  <in>   ;(input)
v3583    pin 2 =  <out>  ;(output)
v3584    pin 7 = gnd
v3585    pin 14 = +5v
v3586endtext
v3587skip 32
v3588com element 2
```

```
v3589begin
v3590    pin 3 =  <in>    ;(input)
v3591    pin 4 =  <out>   ;(output)
v3592endtext
v3593skip 26
v3594com element 3
v3595begin
v3596    pin 5 =  <in>    ;(input)
v3597    pin 6 =  <out>   ;(output)
v3598endtext
v3599skip 20
v3600com element 4
v3601begin
v3602    pin 9 =  <in>    ;(input)
v3603    pin 8 =  <out>   ;(output)
v3604endtext
v3605skip 14
v3606com element 5
v3607begin
v3608    pin 11 =  <in>   ;(input)
v3609    pin 10 =  <out>  ;(output)
v3610endtext
v3611skip 8
v3612com element6
v3613begin
v3614    pin 13 =  <in>   ;(input)
v3615    pin 12 =  <out>  ;(output)
v3616endtext
v3617com
v3618calc isne=0
v3619com
v3620calc isne=isne+1
v3621com******************************************************************
v3622h.nand2     (a,b,o::0,0,0,4,0,3622,3675)
v3623com nand gate, 2 input
v3624com list=input a,b, output1:lat,pwr,chips,calc,incl,addrs
v3625if nbe .ne. 1 skip 4
v3626calc nbn=icn
v3627calc icn=icn+1
v3628attr pwr = pwr + 60
v3629attr chips=chips + 1
v3630begin htext
v3631  ttl nand gate, 2 input
v3632   ic <nbn>, element  <nbe>
v3633endtext
v3634if nbe .eq. 1 skip 4
v3635if nbe .eq. 2 skip 14
v3636if nbe .eq. 3 skip 21
v3637if nbe .eq. 4 skip 28
v3638com element 1
v3639begin htext
v3640   device is sn7400n
```

```
v3641    connections:
v3642      pin  7 = grd
v3643      pin 14 = +5v
v3644      pin  1 =  <a>
v3645      pin  2 =  <b>
v3646      pin  3 =  <o>
v3647endtext
v3648skip 25
v3649com element 2
v3650begin htext
v3651    connections:
v3652      pin  4 =  <a>
v3653      pin  5 =  <b>
v3654      pin  6 =  <c>
v3655endtext
v3656skip 17
v3657com element 3
v3658begin htext
v3659    connections:
v3660      pin  9 =  <a>
v3661      pin 10 =  <b>
v3662      pin  8 =  <c>
v3663endtext
v3664skip 9
v3665com element 4
v3666begin htext
v3667    connections:
v3668      pin 12 =  <a>
v3669      pin 13 =  <b>
v3670      pin 11 =  <c>
v3671endtext
v3672calc nbe = 0
v3673com common end
v3674calc nbe = nbe +1
v3675com*******************************************************************
v3676h.nand3     (a,b,c,o::0,0,0,4,0,3676,3723)
v3677com nand gate, 3 input
v3678com list= inputs a,b,c, output::
v3679if nce .ne. 1 skip 4
v3680calc ncn = icn
v3681calc icn = icn + 1
v3682attr pwr = pwr + 60
v3683attr chips=chips + 1
v3684begin htext
v3685  ttl nand gate, 3 input
v3686   ic <ncn>, element  <nce>
v3687endtext
v3688if nce .eq. 1 skip 3
v3689if nce .eq. 2 skip 14
v3690if nce .eq. 3 skip 21
v3691com element 1
v3692begin htext
```

```
v3693   device is sn7410n
v3694    connections:
v3695      pin  7 = grd
v3696      pin 14 = +5v
v3697      pin  1 =  <a>
v3698      pin  2 =  <b>
v3699      pin 13 =  <c>
v3700      pin 12 =  <o>
v3701endtext
v3702skip 19
v3703com element 2
v3704begin htext
v3705    connections:
v3706      pin  3 =  <a>
v3707      pin  4 =  <b>
v3708      pin  5 =  <c>
v3709      pin  6 =  <o>
v3710endtext
v3711com element 3
v3712begin htext
v3713    connections.
v3714      pin  9 =  <a>
v3715      pin 10 =  <b>
v3716      pin 11 =  <c>
v3717      pin  8 =  <o>
v3718endtext
v3719skip 10
v3720calc nce = 0
v3721com common end
v3722calc nce = nce+1
v3723com*******************************************************************
v3724h.nand4     (a,b,c,d,o::0,0,0,4,0,3724,3762)
v3725com nand gate, 4 input
v3726com list= inputs abcd,output::lat,pwr,chips,calc,incl,addrs
v3727if nde .ne. 1 skip 4
v3728calc ndn = icn
v3729calc icn = icn + 1
v3730attr pwr = pwr + 60
v3731attr chips=chips + 1
v3732begin htext
v3733  ttl nand gate, 4 input
v3734   ic <ndn>, element  <nde>
v3735endtext
v3736if nde .eq. 1 skip 2
v3737if nde .eq. 2 skip 13
v3738com element 1
v3739begin htext
v3740   connections.
v3741      pin  7 = grd
v3742      pin 14 = +5v
v3743      pin  1 =  <a>
v3744      pin  2 =  <b>
```

```
v3745      pin  4 =  <c>
v3746      pin  5 =  <d>
v3747      pin  6 =  <o>
v3748endtext
v3749skip 11
v3750com element 2
v3751begin htext
v3752   connections:
v3753      pin  9 =  <a>
v3754      pin 10 =  <b>
v3755      pin 12 =  <c>
v3756      pin 13 =  <d>
v3757      pin  8 =  <o>
v3758endtext
v3759calc ndn =0
v3760com common end
v3761calc ndn = ndn+1
v3762com********************************************************************
v3763h.nand8     (a,b,c,d,e,f,g,h,o::0,60,1,19,0,3763,3783)
v3764com nand gate, 8 input
v3765com list=inputs abcdefgh,output::lat,pwr,chips,calc,incl,addrs
v3766begin htext
v3767  ttl nand gate, 8 input
v3768   ic <icn>
v3769   connections:
v3770      pin  7 = grd
v3771      pin 14 = +5v
v3772      pin  1 =  <a>
v3773      pin  2 =  <b>
v3774      pin  3 =  <c>
v3775      pin  4 =  <d>
v3776      pin  5 =  <e>
v3777      pin  6 =  <f>
v3778      pin 11 =  <g>
v3779      pin 12 =  <h>
v3780      pin  8 =  <o>
v3781endtext
v3782calc icn = icn + 1
v3783com********************************************************************
v3784h.ffjk      (j,k,q,nq,s,r,ck::0,0,0,4,0,3784,3835)
v3785com ttl flip-flop, jk
v3786com list = j,k,q,qbar,set,rejet,clock::lat,pwr,chips,calc,incl,addr
v3787if jke .ne. 1 skip 4
v3788calc jkn = icn
v3789calc icn = icn + 1
v3790attr pwr = pwr + 60
v3791attr chips=chips + 1
v3792begin htext
v3793 ttl j-k flip-flop
v3794  ic <jkn>, element  <jke>
v3795endtext
v3796if jke .eq. 1 skip 2
```

```
v3797if jke .eq. 2 skip 16
v3798com element 1
v3799begin htext
v3800    device is 8822
v3801     connections:
v3802      pin  8 = grd
v3803      pin 16 = +5v
v3804      pin  4 =  <j>
v3805      pin  7 =  <k>
v3806      pin  6 =  <q>
v3807      pin  5 =  <nq>
v3808      pin  2 =  <s>
v3809      pin  3 =  <r>
v3810      pin  1 =  <ck>
v3811endtext
v3812skip 13
v3813com,element 2
v3814begin htext
v3815     connections:
v3816      pin 15 =  <j>
v3817      pin 11 =  <k>
v3818      pin 10 =  <q>
v3819      pin  9 =  <nq>
v3820      pin 13 =  <s>
v3821      pin 14 =  <r>
v3822      pin 12 =  <ck>
v3823endtext
v3824calc jke = 0
v3825com common end
v3826calc jke = jke + 1
v3827com******************************************************************
v3828com
v3829com******************************************************************
v3830com
v3831com  digital communications realizations (pollock)
v3832com
v3833com******************************************************************
v3834com
v3835com******************************************************************
v3836h.rs232conn (in,sg,out,pg::0,0,0,11,0,3836,3848)
v3837com rs232c i/o connector
v3838com list = input,rtn,out,protgrd:lat,pwr,chips,calc,incl,addrs
v3839begin htext
v3840  standard rs232c connector, j <jn>,for signals  <in>, <out>
v3841     connections:
v3842       pin  3 =  <in>   (received data)
v3843       pin  7 =  <sg>   (signal ground)
v3844       pin  2 =  <out>  (transmitted data)
v3845       pin  1 =  <pg>   (protective ground)
v3846endtext
v3847calc jn=jn+1
v3848com******************************************************************
```

```
v3849h.rs232tx   (in,out::0,0,0,4,0,3849,3893)
v3850com rs232c driver
v3851com list=input, output::lat,pwr,chips,calc,incl,addrs
v3852if rsde .ne. 1 skip 4
v3853calc rsic = icn
v3854calc icn =icn+1
v3855attr pwr = pwr + 150
v3856attr chips = chips + 1
v3857begin htext
v3858  rs232c driver, element  <rsde> of ic <rsic>
v3859    device is fairchild 9616dc triple rs232c line driver
v3860     connections:
v3861endtext
v3862if rsde .eq. 1 skip 4
v3863if rsde .eq. 2 skip 13
v3864if rsde .eq. 3 skip 19
v3865com element 1
v3866begin htext
v3867       pin 7  = gnd
v3868       pin 8  = -15v
v3869       pin 14 = +15v
v3870       pin 1  =  <in>
v3871       pin 2  = true
v3872       pin 3  = gnd    (inhibit)
v3873       pin 4  =  <out>
v3874endtext
v3875skip 16
v3876com element 2
v3877begin htext
v3878       pin 5  =  <in>
v3879       pin 6  = gnd
v3880       pin 10 =  <out>
v3881endtext
v3882skip 9
v3883com element 3
v3884begin htext
v3885       pin 13 =  <in>
v3886       pin 12 = true
v3887       pin 11 = gnd
v3888       pin  9 =  <out>
v3889endtext
v3890calc rsde = 0
v3891com common completion
v3892calc rsde =rsde+1
v3893com********************************************************************
v3894h.rs232rx   (in,out::0,0,0,9,0,3894,3948)
v3895com rs232c receiver
v3896com list=input,output::lat,pwr,chips,calc,incl,addrs
v3897com **
v3898com ** fully commented model for multi-element ic's **
v3899com **
v3900com if the element counter for this ic is 1,synchronize the
```

```
v3901com local ic counter with the master ic counter
v3902if rsre .ne. 1 skip 5
v3903calc icn = icn + 1
v3904calc rsric = icn
v3905com add power and chip counts to globals
v3906calc pwr = pwr + 150
v3907calc chips=chips+1
v3908com print common heading
v3909begin htext
v3910  rs232c receiver, element  <rsre> of ic <rsric>
v3911   device is fairchild 9617dc triple rs232c line driver
v3912    connections:
v3913endtext
v3914com branch on element number
v3915if rsre .eq. 1 skip 3
v3916if rsre .eq. 2 skip 14
v3917if rsre .eq. 3 skip 22
v3918com element 1 .  also pick up power and common signals
v3919begin htext
v3920      pin  7 = gnd
v3921      pin 14 = +5v
v3922      pin  4 =  <in>  ;(input)
v3923      pin  3 = n.c.   (response)
v3924      pin  2 = n.c.   (hysteresis)
v3925      pin  1 =  <out> ;(output)
v3926endtext
v3927com skip to end
v3928skip 17
v3929com element 2
v3930begin htext
v3931      pin 10 =  <in>  ;(input)
v3932      pin 11 = n.c.   (response)
v3933      pin 12 = n.c.   (hysteresis)
v3934      pin 13 =  <out> ;(output)
v3935endtext
v3936skip 9
v3937com element 3 . reset element counter
v3938begin htext
v3939      pin  9 =  <in>  ;(input)
v3940      pin  8 = n.c.   (response)
v3941      pin  6 = n.c.   (hysteresis)
v3942      pin  5 =  <out> ;(output)
v3943endtext
v3944calc rsre = 0
v3945com common conclusion increments element counter
v3946calc rsre = rsre + 1
v3947com end of realization
v3948com*******************************************************************
v3949h.uart      (si,so,rc,tc::0,500,1,3,0,3949,3998)
v3950com primitive to define uart
v3951com list = input,output,rclock,tclock::lat,pwr,chips,calc,incl,addr
v3952 calc inport=inport+1
```

```
v3953calc outprt=outprt+1
v3954begin htext
v3955  universal assynchronous receiver/transmitter (uart) ic <icn>
v3956   device is general instrument tr1602a
v3957    connections:
v3958      pin  1 = +5v
v3959      pin  2 = -12v
v3960      pin  3 = grd
v3961      pin  4 = .not. (decode a(0:7) value  <inport> .and.
v3962      pin             inp .and. dbin)  (rde)
v3963      pin  5-12 = db(8:1)    (input data)
v3964      pin 13 = db(1)         (pe)
v3965      pin 14 = db(2)         (fe)
v3966      pin 15 = db(3)         (or)
v3967      pin 16 = .not. (decode a(0:7) value  <inport>+1 .and.
v3968      pin             inp .and. dbin   (swe)
v3969      pin 17 =  <rc>          (rcp)
v3970      pin 18 jumper to 4     (rda)
v3971      pin 19 = db(4)         (da)
v3972      pin 20 =  <si>          (si)
v3973      pin 21 = reset         (xr)
v3974      pin 22 = db(5)         (tmbt)
v3975      pin 23 = .not. (decode a(0:7) value  <outprt> .and.
v3976                     out .and..not. wr-bar) (ds)
v3977      pin 24 = n.c.          (eoc)
v3978      pin 25 =  <so>          (so)
v3979      pin 26-33 = db(8:1)    (output data)
v3980      pin 34 = true          (cs)
v3981      pin 35 = gnd           (np)
v3982      pin 36 = gnd           (tsb)
v3983      pin 37 = true          (n2)
v3984      pin 38 = true          (nb1)
v3985      pin 39 = gnd           (eps)
v3986      pin 40 =  <tc>          (tcp)
v3987endtext
v3988calc icn=icn+1
v3989calc inport=inport+1
v3990com******************************************************************
v3991com
v3992com******************************************************************
v3993com
v3994com  discrete component realizations (pollock)
v3995com
v3996com******************************************************************
v3997com
v3998com******************************************************************
v3999h.resmfqtrwt(sigin,sigout,ohms:1,2000000:0,0,0,9,0,3999,4009)
v4000com primitive to define 1/4 watt metal film 1% resistor
v4001com list=input signal,output signal:min/max ohms:
v4002com      lat,pwr,chips,calc,incl,addrs
v4003begin htext
v4004  resistor r <rn>,  <ohms> ohms, 1/4 watt 1% metal film
```

```
v4005        pin 1 =  <sigin>
v4006        pin 2 =  <sigout>
v4007endtext
v4008calc rn=rn+1
v4009com********************************************************************
v4010h.rpack-18b (in,out::0,0,0,4,0,4010,4078)
v4011com primitive to define 180 ohm resistor pack, 8 res. in 16 pin dip
v4012com list=input,output::
v4013if  <rabe> .ne. 1 skip 2
v4014calc  <r18n> = rpn
v4015calc rpn=rpn+1
v4016begin htext
v4017  resistor r <rn>, 180 ohms, element  <rabe> of resistor pack  <r18n>
v4018endtext
v4019if  <rabe> .eq. 1 skip 8
v4020if  <rabe> .eq. 2 skip 10
v4021if  <rabe> .eq. 3 skip 20
v4022if  <rabe> .eq. 4 skip 20
v4023if  <rabe> .eq. 5 skip 30
v4024if  <rabe> .eq. 6 skip 30
v4025if  <rabe> .eq. 7 skip 40
v4026if  <rabe> .eq. 8 skip 50
v4027com element 1
v4028begin htext
v4029   pin 1 =  <in>
v4030   pin 16 =  <out>
v4031endtext
v4032skip 43
v4033com element 2
v4034begin htext
v4035   pin 2 =  <in>
v4036   pin 15 =  <out>
v4037endtext
v4038skip 37
v4039com element 3
v4040begin htext
v4041   pin 3 =  <in>
v4042   pin 14 =  <out>
v4043endtext
v4044skip 31
v4045com element 4
v4046begin htext
v4047   pin 4 =  <in>
v4048   pin 13 =  <out>
v4049endtext
v4050skip 25
v4051com element 5
v4052begin htext
v4053   pin 5 =  <in>
v4054   pin 12 =  <out>
v4055endtext
v4056skip 19
```

```
v4057com element 6
v4058begin htext
v4059   pin 6 =  <in>
v4060   pin 11 =  <out>
v4061endtext
v4062skip 13
v4063com element 9
v4064begin htext
v4065   pin 7 = in>
v4066   pin 10 =  <out>
v4067end htext
v4068skip 5
v4069com element 10
v4070begin htext
v4071   pin 8 =  <in>
v4072   pin 9 =  <out>
v4073endtext
v4074calc rabe = 0
v4075com common end
v4076calc rabe = rabe + 1
v4077calc rn = rn + 1
v4078com******************************************************************
v4079h.trimpot   (in,out,w,r:50,2000000:0,0,0,10,0,4079,4090)
v4080com primitive to define trim pot
v4081com list = input signal, output signal, wiper signal, ohms:
v4082com        res range:lat,pwr,chips,calc,incl,addrs
v4083begin htext
v4084  trimpot, r <rn>,  <r> ohms, 22t 1/2w cermep
v4085    pin 1 =  <in>
v4086    pin 2 =  <w>     (wiper)
v4087    pin 3 =  <out>   (cw end)
v4088endtext
v4089calc rn=rn+1
v4090com******************************************************************
v4091h.capac-cer (in,out,val:10,9900000:0,0,0,10,0,4091,4102)
v4092com primitive to define ceramic capacitor
v4093com list=input signal,output signal,value:value range (pf):
v4094com       lat,pwr,chips,calc,incl,addrs
v4095begin htext
v4096  capacitor, c <cn>, ceramic,  <val> pf
v4097    connections:
v4098      pin 1 =  <in>
v4099      pin 2 =  <out>
v4100endtext
v4101calc cn=cn+1
v4102com******************************************************************
v4103h.diode-sw  (sigpos,signeg::0,0,0,10,0,4103,4114)
v4104com primitive to define a silicon switching diode
v4105com list = positive signal, negative signal::lat,pwr,chips,c,i,a@
v4106begin htext
v4107  dioce cr <crn>, silicon switching type
v4108  device is 1n4447
```

```
v4109    connections:
v4110      pin 1 =  <sigpos>  (anode)
v4111      pin 2 =  <signeg>  (cathode)
v4112endtext
v4113calc crn=crn+1
v4114com*******************************************************************
v4115h.diode-znr (sigpos,signeg,v:1,200:0,0,0,13,0,4115,4129)
v4116com primitive to defina a zener diode, 1/4 watt
v4117com list=positive,negative ends, value:value range:
v4118com      lat,pwr,chips,c,i,a
v4119begin htext
v4120  zener diode, cr<crn>
v4121    device is 1/4 watt zener diode, zener voltage <v>
v4122      connections:
v4123         pin  1 = <sigpos>   (cathode of diode)
v4124         pin  2 = <signeg>   (anode)
v4125      note that positive signal is connected to cathode for
v4126      zener operaton.
v4127endtext
v4128calc crn=crn+1
v4129com*******************************************************************
v4130h.conn-al   (in,ret,shld,name::0,0,0,11,0,4130,4142)
v4131com primitive to define dip socket used as an analog connector
v4132com list=input, return, shield, name::time,pwr,chips,c,i,a
v4133begin htext
v4134   connector j <jn>, for analog signal  <name>,
v4135    16 pin dip socket
v4136     connections:
v4137       pin 1 =  <in>
v4138       pin 2 =  <ret>
v4139       pin 3 =  <shld>  grounded at signal source only
v4140endtext
v4141calc jn=jn+1
v4142com*******************************************************************
v4143s.anain     (sig,h,l,b:0,8,50,-25,1,100:0,0,0,0,11,4143,4174)
v4144com primitive to define processor controlled analog input
v4145com list = input,hi volt limit,lo volt limit,3db rolloff:
v4146com        bits,voltage limits,bw limits:
v4147com        time,stor,ext,calc,incl,addrs
v4148com added by m.r. heilstedt, may 1983
v4149name $na001-$na030
v4150begin htext
v4151  ;analog input channel for signal <sig>, range <h> to <l> volts,
v4152;  3db rolloff at <b> khz
v4153endtext
v4154incl h.conn-al   (<$na006>,<$na007>,gnd,<sig>::)
v4155com select gain for buffer amp to match input range
v4156if <l> .lt.  0 skip 4
v4157if <h> .le. 10 skip 6
v4158if <h> .le. 25 skip 8
v4159if <h> .le. 50 skip 10
v4160com set buffer amp if bipolar output
```

```
v4161if <l> .ge.  -5 skip 3
v4162if <l> .ge. -12 skip 5
v4163if <l> .eq. -25 skip 7
v4164com gain 1.0 (expressed 10) for input range 0,+10 volts;+-5 volts
v4165incl h.bufframp  (<$na006>,<$na007>,<$na005>,10,<b>::)
v4166skip 5
v4167com gain 2.5 (expressed 25) for input range 0,+25 volts;+-12.5 volts
v4168incl h.bufframp  (<$na006>,<$na007>,<$na005>,25,<b>::)
v4169skip 2
v4170com gain 5.0 (expressed 50) for input range 0,+50 volts;+-25 volts
v4171incl h.bufframp  (<$na006>,<$na007>,<$na005>,50,<b>::)
v4172incl h.adc2      (<$na005>,<sig>,<h>,<l>:8:)
v4173call s.sensecond (<sig>:8,128)
v4174com******************************************************************
v4175h.adc2      (in,out,h,l:0,8:0,800,1,50,7,4175,4226)
v4176com primitive to define 8 bit processor controlled adc
v4177com list = input,output,hi volt limit, lo volt limit:
v4178com        bits:lat,pwr,chips,calc,incl,addrs
v4179com added by m. r. heilstedt, may 1983
v4180name $na031-$na060
v4181com add resistor trimmer in series with input
v4182incl h.trimpot   (gnd,<$na041>,<in>,200:)
v4183begin htext
v4184   a/d converter, 8 bit
v4185      device is analog devices ad570, ic <icn>
v4186         connections:
v4187           pin  1 = n.c.
v4188           pin  2 = <out>(0)  (lsb)
v4189           pin  3 = <out>(1)  (2lsb)
v4190           pin  4 = <out>(2)  (3lsb)
v4191           pin  5 = <out>(3)  (4lsb)
v4192           pin  6 = <out>(4)  (5lsb)
v4193           pin  7 = <out>(5)  (6lsb)
v4194           pin  8 = <out>(6)  (7lsb)
v4195           pin  9 = <out>(7)  (msb)
v4196           pin 10 = +5 volts
v4197           pin 11 = conv      (blank and .not. convert)
v4198           pin 12 = -15 volts
v4199           pin 13 = <$na041>  (analog input)
v4200           pin 14 = gnd       (analog common)
v4201endtext
v4202if <l> .lt. 0 skip 6
v4203com if unipolar input, make direct connections
v4204begin htext
v4205           pin 15 = gnd       (bipolar offset)
v4206           pin 16 = gnd       (digital common)
v4207endtext
v4208skip 11
v4209com if bipolar input, add ttl interface
v4210begin htext
v4211           pin 15 = <$na049>  (bipolar offset)
v4212           pin 16 = <$na048>  (digital common)
```

```
v4213endtext
v4214incl h.nand4     (0,1,+5v,<$na048>,<$na045::)
v4215incl h.invert    (<$na045>,<$na046>::)
v4216incl h.diode-sw  (<$na046>,<$na047>::)
v4217incl h.diode-sw  (<$na047>,<$na048>::)
v4218incl h.diode-sw  (<$na048>,<$na049>::)
v4219incl h.resmfqtrwt(-15v,<$na048>,30000:)
v4220com finish connections
v4221begin htext
v4222           pin 17 = dr        (data ready)
v4223           pin 18 = n.c.
v4224endtext
v4225calc icn=icn+1
v4226com
```

LIST OF REFERENCES

1. Daniel, M.E. and Gwyn, C.W., "CAD Systems for IC Design", IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, v. CAD-1, pp. 2-3, Jan 1982.

2. Matelan, M.N., "Automating the Design of Dedicated Real Time Control Systems", Preprint UCRL-78651, Lawrence Livermore Laboratories, 21 August 1976.

3. Heilstedt, M.R., Automated Design of Microprocessor-Based Digital Filters, MS Thesis, Naval Postgraduate School, Monterey, California, p. 23, 1983.

4. Ross. A.A., Computer Aided Design of Microprocessor-Based Controllers, Ph.D. Thesis, University of California, Davis, 1978.

5. Carson, T.H., An Input Translator for a Computer Aided Design System, MS Thesis, Naval Postgraduate School, Monterey, California, 1984.

6. Smith, T.J., Implementation of a Zilog Z-80 Based Realization Library for the Control System Design Environment, MS Thesis, Naval Postgraduate School, Monterey, California, 1984.

7. Pollock, G.G., Further Development and Investigation of Computer-Aided Design of Microprocessor Systems, MS Thesis, University of California, Davis, 1980.

8. iAPX 86/88,186/188 User's Manual, v. 2, p. 6-5, Intel Corporation, 1983.

9. Ibid., p. 6-3.

10. Ibid., p. 6-18.

11. Ibid., p. 4-5.

12. Ibid., p. 6-6.

13. Statz, R.J., 8087, Prentice-Hall, 1983.

14. Kern, C.O., "Five C Compilers for CP/M-80", Byte, v. 8, pp. 123-124, 1983.

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center 2
Cameron Station
Alexandria, Virginia 22314

2. Library, Code 0142 2
Naval Postgraduate School
Monterey, California 93943

3. Professor H. H. Loomis, Jr., Code 62Lm 4
Department of Electrical and
Computer Engineering
Naval Postgraduate School
Monterey, California 93943

4. Lieutenant Colonel A. A. Ross, USAF, Code 52Rs 2
Department of Computer Science
Naval Postgraduate School
Monterey, California 93943

5. LCDR Alan Cetel, USN 2
Naval Sea Systems Command Headquarters
Washington, D.C. 20362

208304

Thesis
C33842 Cetel
c.1
Implementation of an intel 8086 microprocessor-based realization library for the Control System Design Language.